A Virtualization Approach to Scalable Enterprise Content Management

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik
der Universität Stuttgart
zur Erlangung der Würde eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte Abhandlung

Vorgelegt von

Frank Oliver Wagner
aus Stuttgart

Hauptberichter:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Mitberichter:	Prof. Dr.-Ing. Norbert Ritter
Tag der mündlichen Prüfung:	7. März 2011

Institut für Parallele und Verteilte Systeme (IPVS)
der Universität Stuttgart

2011

Preface

This work evolved while I was working at the department Applications of Parallel and Distributed Systems at the Institute of Parallel and Distributed Systems at the University of Stuttgart. I would like to thank everybody who contributed to this work and supported it.

First of all, I want to thank Prof. Bernhard Mitschang for the supervision and survey of the thesis. I also want to thank him for the discussions, the support and the guidance over the years. And I want to thank him for giving me the opportunity to work on an interesting topic with practical relevance.

Furthermore, I want to thank Prof. Norbert Ritter for co-reviewing the thesis, and also for his support during the project. In this context I also want to thank his research assistant Kathleen Krebs for the discussions and the practical work before, during and after the common project.

The work was partially sponsored by Centers of Advanced Studies and Shared University Research Grants from the IBM Deutschland Research and Development GmbH. I am grateful for this support and especially want to thank Cataldo Mega for his engagement in organizing the funding. I also want to thank him for the discussions which largely influenced this thesis. I also want to thank all the IBMers that supported the thesis, especially Stewart Tate, Patrick Dantressangle, Adrian Lee, Michael Bauer, Dieter Schieber and Udo Hertz.

I also want to thank my colleagues at the University of Stuttgart for many interesting discussions and the good working atmosphere. I would especially like to thank Alexander Moosbrugger for his support in the project and for proof-reading the thesis. Furthermore, I want to thank Tobias Kraft, Christoph Mangold, Marcello Mariucci, Oliver Schiller, Marko Vrhovnik and Ralf Wagner.

Many thanks also go to all the students at the University of Stuttgart and the University of Hamburg that were working on the prototype as part of a thesis or an internship: Kenan Bahcivan, Malte Biss, Christian Ewers, Peter Gruber, Othello Maurer, Alexander Moosbrugger, Saleh Mohammad Pour, Martin Russold, Sergej Schütz and Tim Waizenegger.

Last but not least, I want to thank my family for their continuous support and encouragement during the studies and the work on this thesis.

Contents

List of Figures

Abbreviations

ACID	Atomicity, Consistency, Isolation, Durability
AIIM	Association for Information and Image Management
API	Application Programming Interface
CD	Compact Disc
CFS	Cluster File System
CM	IBM DB2 Content Manager
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CRUD	Create, Retrieve, Update, Delete
DBMS	DataBase Management System
DHT	Distributed Hash Table
DVD	Digital Versatile Disc
EADM	E-Mail Archiving, Discovery and Management
ECM	Enterprise Content Management
FS	File System
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IMAP	Internet Message Access Protocol
LDAP	Lightweight Directory Access Protocol
LS	Library Server
LUN	Logical UNit (in a SAN)
LVM	Logical Volume Manager
MIME	Multipurpose Internet Mail Extensions
NAS	Network Attached Storage
NUMA	Non-Uniform Memory Architecture
OGSA	Open Grid Services Architecture
OGSI	Open Grid Services Infrastructure
OLTP	OnLine Transaction Processing
P2P	Peer-to-Peer
PDF	Portable Document Format

POP	Post Office Protocol
RAID	Redundant Array of Inexpensive Disks
RFC	Request for Comments
RM	Resource Manager
RMI	Remote Method Invocation
RVL	Repository Virtualization Layer
SAN	Storage Area Network
SATA	Serial Advanced Technology Attachment
SIS	Single-Instance Storage
SMP	Symmetrical Multi Processors
SMTP	Simple Mail Transfer Protocol
SOA	Service-Oriented Architecture
SOX	Sarbanes-Oxley Act
UI	User Interface
UTF	Unicode Transformation Format
WCM	Web Content Management
XML	Extensible Markup Language

Abstract

In this dissertation we present an approach for the implementation of a scalable content management system that is based on the virtualization of the repository. It allows to dynamically scale-out the repository on multiple machines in order to adjust the system to the current load. This is an important precondition when offering a content management system as a service in a cloud to customers of different and changing sizes. As an example we will look at e-mail archiving.

In personal and especially in business life one is daily confronted with a huge amount of information. They have to be processed and archived according to their importance. Yet, their importance may change over time. Information classified as not very important when it arrives may become essential in the future. Thus an effective and efficient management of all information is necessary.

Information can be in different forms from highly structured over semi-structured to completely unstructured information. According to different sources the semi-structured and unstructured information makes up between 80% and 85% percent of all business information. Examples are text documents, e-mails and short messages exchanged between colleagues, business partners, suppliers and customers. The appropriate management of this information is very important. It has to support the typical use cases as well as huge and changing volumes of information, and it has to be cost-efficient. Moreover, in a globalized world the information has to be accessible from anywhere on the world at any time.

E-mails have become a very important tool for the transportation of information within and between enterprises. Often the e-mails with their information remain in the e-mail system. Thus the e-mail system has become an important pool of information and also of knowledge. Furthermore, by analyzing sets of e-mails for a topic from different users it is often possible to determine how a process evolved and why a specific decision was made. This makes the e-mail system also a very important tool for the post-mortem analysis of an incident.

The data volumes an e-mail system has to deal with are very high. Thus setting up an e-mail system is a challenging task and requires accurate planning. Yet, in today's dynamic business environment the requirements may change quickly. Once the e-mails are locked into an inappropriate system it will be very expensive to migrate them into a new system. Thus a flexible system that can be dynamically adjusted to the current situation is advantageous.

In this dissertation we present an approach that allows growing and shrinking the system in a wide range. Therefore we first introduce a repository virtualization layer that decouples the applications from the direct usage of the repository. For each request it determines the responsible repository and forwards the request to it. Transparent for the applications and their users it is possible to add new machines to the system or to remove superfluous ones.

First we describe the data an e-mail archive has to deal with. These are first and foremost the e-mails, but there are some other metadata, too. For the implementation of the data model we are using a combination of a search engine and a relational database. The focus lies on the search engine as it is better suited for semi-structured and unstructured information. With the typical fuzzy information need of the user the search engine provides more relevant results than the rather strict and precise databases. When ingesting a new e-mail into the system the full-text is extracted and is added to a full-text index. A cluster of search engines makes the full-text indexes accessible by the users. The database is mostly used for consistency information and access control information. A normalization of the information in the search engine is also investigated. When e-mails are sent to a larger group or when they are forwarded several times there are many partial duplicates. Therefore we indexed the attachments of the e-mail separated from its header and body. But this turned out to be completely impractical during search. Thus now each e-mail is added completely to the full-text index.

The repository virtualization layer allows distributing the data over multiple machines. The starting point for this distribution is the partitioning schema. In order to achieve a good performance a good scalability is important to keep the communication between the machines at a minimum. Especially very frequent operations and time-critical interactive operations should be executed locally on one machine and join operations should be avoided. For the distribution of the data to the machines we are using a distributed hash table from the peer-to-peer system OpenChord. By using consistent hashing the identifiers of the documents, which we compute as a hash over their content, are uniquely mapped to the currently responsible machine.

To store the original documents we evaluated different approaches. In the simplest approach a regular file system was chosen. It has the lowest overhead, but also provides the fewest optional features. Thus e.g. to rearrange the data, one has to implement a different mechanism which typically by transmits them over the network. Shared file systems like network file systems and cluster file systems have an advantage here. With an appropriate organization of the file system it is relatively easy and fast to assign even large fractions of the documents to another machine. When the documents are stored in a content management system further functionalities provided by these systems like replication and hierarchical storage management can be exploited. We are also looking at peer-to-peer systems which provide a wide spectrum of different functionalities.

Finally, we discuss the implementation of the repository virtualization layer. Besides forwarding the operations on individual documents to the responsible machine, the repository virtualization layer also has to rearrange the data in case a machine joins or leaves the system, or in order to balance the load. While moving data from one machine to another this fraction of the data will not be available. In order to keep this time low the movement is done in two phases: In the first phase only the data that is absolutely necessary for the new node to start working is transferred. In the second phase, where the larger fraction of the data is moved, no locking is necessary. This is implemented as an extension to OpenChord that deals with some additional operations and the movement of the large data sets.

In this work the focus is on the technical aspects that are necessary when implementing a flexible and scalable content management system. The concrete example is a system for e-mail archiving. It allows adjusting the system to the current processing and storage requirements. This is an important prerequisite when offering e-mail archiving as a service in a cloud environment. Further aspects like setting up the system, monitoring and adjusting the system and billing are topic of a thesis actually done at the University of Hamburg.

Zusammenfassung[1]

In dieser Arbeit wird ein Ansatz zur Implementierung eines skalierbaren Content Management Systems vorgeschlagen, dessen Grundlage die Virtualisierung des Repositories ist. Damit wird ein dynamisches scale-out des Repositories auf mehrere Maschinen zur Anpassung an die aktuelle Last möglich. Dies ist eine wichtige Voraussetzung, um ein Content Management System unterschiedlich großen Kunden als Dienstleistung in einer Cloud anbieten zu können. Als Beispielanwendung dient die E-Mail-Archivierung.

Im Alltag, und insbesondere im geschäftlichen Alltag, wird man täglich mit einer großen Menge an Informationen konfrontiert. Diese müssen verwaltet und je nach Wichtigkeit abgelegt werden. Ihre Wichtigkeit kann sich jedoch mit der Zeit noch ändern. Eine aktuell als unwichtig eingestufte Information kann in der Zukunft entscheidend werden. Daher ist deren effektive und effiziente Verwaltung eine wichtige Voraussetzung, um jederzeit schnell an benötigte Informationen zu gelangen, und aus ihnen Wert zu schöpfen.

Informationen treten in unterschiedlichen Formen auf. Stark strukturierte Informationen gibt es z.B. häufig bei der Verarbeitung von Geschäftsprozessen in OLTP Anwendungen. Diese werden meist in relationalen Datenbanken gespeichert. Sehr wichtig im Geschäftsleben sind aber auch semi-strukturierte und unstrukturierte Informationen: Dokumente, E-Mails und Kurznachrichten, die zwischen Kollegen und Geschäftspartnern ausgetauscht werden, um Informationen weiter zu geben, Aktivitäten zu koordinieren und Entscheidungen zu kommunizieren. Diese machen einen großen Anteil der Daten in einem Unternehmen aus. Nach verschiedenen Studien fallen etwa 80% bis 85% der geschäftlichen Informationen in diese zweite Kategorie. Die Verwaltung der Informationen muss die funktionalen Anforderungen der verschiedenen Nutzer möglichst gut und vollständig umsetzen, und sie muss für die Verarbeitung der großen und schwankenden Datenmengen auch effizient und skalierbar sein. Daneben ist die Wirtschaftlichkeit sehr wichtig, in die auch die Kosten für Anschaffung, Implementierung, Betrieb und Wartung eingehen.

[1] A summary of the dissertation in German

Große Unternehmen verwenden seit vielen Jahren IT-Systeme um Inhalte zu erfassen, zu verwalten, zu speichern und aufzubewahren. Inhalte können dabei Daten, Metadaten, Dokumente, Web-Seiten und vieles mehr sein. Zu deren Verwaltung sind unterschiedliche Arten von Content Management Systemen entstanden.

Der Kern von Enterprise Content Management (ECM) ist die Verwaltung und Nutzung der unterschiedlichen Inhalte über die Grenzen von einzelnen Anwendungen und Geschäftsprozessen hinweg. Die grundlegende Funktionalität von ECM ist es, komplexen Geschäftsprozessen den Zugriff auf relevante Informationen an verteilten Aufbewahrungsorten zu ermöglichen. Z.B. benötigt ein Angestellter einer Versicherung bei der Bearbeitung eines Falls Zugriff auf Informationen von der Versicherungspolice des Kunden bis zu Polizeiberichten und Beweisfotos zum aktuellen Fall. Wenn diese Informationen nicht direkt verfügbar sind, dann muss der Angestellte sie erst beschaffen. Damit steigen natürlich die Bearbeitungszeit und die Kosten für den Fall. Ein anderes Beispiel sind Aktienhändler, die auf Grund von Gesetzen und sonstigen Regulierungen die ausgetauschten Informationen, und dazu gehören auch alle E-Mails, archivieren müssen. In großen Unternehmen müssen so täglich Millionen von E-Mails entgegengenommen, analysiert, klassifiziert und archiviert werden. Dort müssen sie dann für mehrere Jahre revisionssicher aufbewahrt werden und auch jederzeit gefunden und zurückgeholt werden können.

Heutzutage, in einer globalisierten Welt, müssen viele Unternehmen den Zugriff auf ihre wichtigsten Daten rund um die Uhr und von überall her ermöglichen. Für die Durchführung der Geschäftsprozesse mit Kunden, Lieferanten und Partnern in aller Welt werden diese Daten stets zeitnah benötigt.

E-Mails sind ein sehr wichtiges Mittel zum Transport von Informationen geworden. Verschiedene Studien haben das aktuelle E-Mail-Aufkommen analysiert und Prognosen für die Zukunft erstellt. Ein Beispiel ist die Studie der Radicati Group aus dem Jahre 2005 [Rad05]. Sie haben 133 E-Mails pro Benutzer und Tag gezählt mit einer durchschnittlichen Größe von 110 KB. Für das Jahr 2009 prognostizierten sie einen Anstieg auf 160 E-Mails pro Tag. Auch andere Studien prognostizieren einen starken Anstieg der Anzahl der E-Mails, aber auch eine zunehmende durchschnittliche Größe der E-Mails.

Wenn die Informationen erst ein Mal im E-Mail-System sind, dann bleiben sie dort auch oft. Das E-Mail-System ist damit auch zu einem großen Informationsdepot geworden. Viele Dokumente mit Informationen sind darin enthalten, und durch die Analyse aufeinanderfolgender E-Mails kann auch der Ablauf von Prozessen zu einem guten Teil nachvollzogen werden. Das E-Mail-System ist somit eine wertvolle Wissensbasis, und es ist eine wichtige Quelle bei der nachträglichen Untersuchung von Abläufen.

Die großen Datenmengen zu beherrschen ist eine anspruchsvolle Aufgabe, die eine gute Planung voraussetzt. In einem sehr dynamischen wirtschaftlichen Umfeld können sich die Anforderungen aber auch schnell wieder ändern. Sind die Daten dann in einem nicht mehr angemessenen System gefangen, dann können sie nur mit sehr hohem Aufwand wieder in ein anderes System migriert werden. Daher ist ein flexibles und dynamisch anpassbares System mit einer guten Skalierbarkeit zu bevorzugen.

In dieser Arbeit wird ein Ansatz vorgeschlagen, der das Wachsen und Schrumpfen des Systems in einem großen Bereich unterstützt. Dazu wird zunächst eine Virtualisierungsschicht eingeführt, die die Anwendungen von der direkten Nutzung des Repositories trennt. Dadurch wird es möglich die Anfragen an die jeweils betroffene Maschine weiter zu leiten. Transparent für den Benutzer können so weitere Maschinen hinzugefügt oder überflüssige wieder entfernt werden.

Zunächst werden die in einem E-Mail-Archiv anfallenden Daten beschrieben und ein Datenmodell aufgestellt. Bei der Implementierung dieses Datenmodells wird auf eine Kombination aus Datenbank und Suchmaschine gesetzt. Dabei liegt der Schwerpunkt hier bei der Suchmaschine, da sie für typischerweise nicht sehr präzisen Suchen auf schwach strukturierten Daten besser geeignet ist. Beim Einfügen von neuen Dokumenten wird der enthaltene Text extrahiert und in einen Volltext-Index eingefügt. Ein Cluster aus Suchmaschinen stellt dem Benutzer dann die Such-Funktionalität auf den erzeugten Volltext-Indexen zur Verfügung. Die Datenbank wird im Wesentlichen für interne Informationen zur Sicherstellung der Konsistenz und für die Zugriffskontrolle verwendet. Eine Normalisierung der Daten in der Suchmaschine zur Reduktion der häufig zu erwartenden Duplikate von Teil-Dokumenten, hier vor allem der Anhänge an den E-Mails, hat sich als nicht praktikabel herausgestellt, so dass jedes Dokument komplett in der Suchmaschine abgelegt werden muss.

Innerhalb der Virtualisierungsschicht können die Daten auf mehrere Maschinen verteilt werden. Die Grundlage dafür bildet ein geeignetes Partitionierungsschema. Eine für die Performanz und Skalierbarkeit des Systems wichtige Bedingung ist eine minimale Kommunikation zwischen den Maschinen. Daher wird hier Wert darauf gelegt, dass häufig ausgeführte sowie zeitkritische, interaktive Operationen möglichst lokal auf einer Maschine durchgeführt werden. Insbesondere sind Verbund-Operatoren (Joins) zwischen großen Datenmengen auf unterschiedlichen Maschinen zu vermeiden. Für die Verteilung der Daten auf die Maschinen wird eine verteilte Hashtabelle aus dem Bereich Peer-to-Peer-Computing verwendet. Diese nimmt den als Prüfsumme über den Inhalt berechneten Schlüssel eines Dokumentes und weist ihn zu jeder Zeit eindeutig einer Maschine zu.

Für die Ablage der rohen Archivdaten werden verschiedene Alternativen untersucht. Die Ablage in einem Dateisystem hat den geringsten Overhead, sie bietet

aber auch die wenigsten zusätzlichen Funktionen. So muss insbesondere für die (Um-)Verteilung der Daten ein anderer Mechanismus gesucht werden, der die Daten dann typischerweise über das Netzwerk kopiert. Hier bieten verteilte Dateisysteme wie z.B. Netzwerk-Dateisysteme und Cluster-Dateisysteme einen Vorteil. Bei einer entsprechenden Organisation der Dateien können auch große Datenmengen relativ einfach einer anderen Maschine zugeordnet werden. Bei der Ablage in einem Content Management System können dort vorhandene Funktionen wie Replikation und hierarchisches Speichermanagement genutzt werden. Auch P2P-Systeme werden betrachtet, die von sich aus eine sehr große Palette an unterschiedlichen Funktionen bieten.

Schließlich wird die Implementierung der Virtualisierungsschicht diskutiert. Neben der Weiterleitung der Operationen auf einzelnen Dokumenten (CRUD-Operationen) ist eine wesentliche Funktion dieser Schicht die Umorganisation der Daten beim Hinzufügen und Entfernen von Maschinen sowie zur Sicherstellung einer gleichmäßigen Verteilung der Last. Während die Daten auf eine andere Maschine verschoben werden stehen diese für einen kurzen Zeitraum nicht zur Verfügung. Um diesen Zeitraum so kurz wie möglich zu halten wird das Verschieben in zwei Phasen aufgeteilt: In der ersten Phase, in der der Bereich der Daten gesperrt wird, werden nur die für die Arbeit absolut notwendigen Daten verschoben. Der weitaus größere Teil der Daten wird in einer zweiten Phase verschoben, in der keine Sperren mehr notwendig sind. Die Implementierung der Virtualisierungsschicht baut auf dem P2P-System OpenChord auf, das per konsistentem Hashing über den Inhalt der Dokumente diese auf die vorhandenen Maschinen verteilt.

In dieser Arbeit liegt der Schwerpunkt auf eher technischen Aspekten bei der Implementierung eines flexiblen und skalierenden Content Management Systems am Beispiel eines Systems zur E-Mail-Archivierung. Es bietet die Möglichkeit das System an die aktuellen Anforderungen bezüglich der Verarbeitungskapazität und der Datenmengen anzupassen. Dies ist eine wichtige Voraussetzung um die E-Mail-Archivierung als Dienst (Application as a Service) in einer Cloud anzubieten. Weitere dabei relevante Aspekte wie z.B. der Aufbau des Systems, die Überwachung und Anpassung des Systems entsprechend der mit dem Kunden vereinbarten Dienstgüte sowie die Abrechnung sind Gegenstand einer aktuell an der Universität Hamburg durchgeführten Arbeit.

1
Introduction

In real life, and especially in business life, one is confronted with a large amount of information. Some of this information is very important, and other information is not so important. Yet, even the currently unimportant information may become important at a later point in time. Therefore an efficient management of the information is necessary to retrieve the information quickly when needed, and to gain value from it.

Information comes in different forms. Structured information like the data created and maintained by an online transaction processing system is often stored in a relational database. Semi-structured and unstructured information like text documents, e-mails, images and videos are also an important kind. According to different fuzzy sources around 80% to 85% of the business information belongs into this second category. Thus appropriate technology to gain knowledge from this information is important.

Large enterprises use software to capture, manage, store, preserve and deliver content for more than 20 years. Content in this case stands for any kind of electronic artifacts: records, data, metadata, documents, websites, etc. that are related to organizational processes. This variety of heterogeneous content types involves various application areas of Enterprise Content Management (ECM).

The main purpose for ECM systems is to manage and leverage any type of information across multiple applications and business processes. Their core functionality is facilitating the access to all the relevant information needed by complex business transactions from their respective content repositories. As an example, a clerk of an insurance company processing a claim needs access to many documents, from the insurance policy of the customer to police reports and appraisal reports for the current claim. When some of this information is not available the clerk has to spend time to get the information and this extends claim processing time and costs. Another example rises from the fact that in today's regulated market all the e-mails flowing in and out of a Stock Broker Company must be monitored and archived due to compliance reasons. Because of this, in a large company up to tens of millions of e-mails and their attachments must be captured, parsed, classified and archived during a business day. This substantial amount of files must be managed and processed during a day, independent of type, size and location.

Today, almost every company faces the need to facilitate access to its key business data anytime and anywhere. The information lifecycle implied by the underlying business models typically requires the use of Enterprise Content Management systems (ECM systems) that allow both, business to business, and business to consumer applications access to content assets to an indefinite number of users around the clock, 7 days a week, 365 days a year.

The Association for Information and Image Management (AIIM), an international non-profit organization representing the content management industry from users to suppliers, defines ECM as follows[1]:

> Enterprise Content Management is the strategies, methods and tools used to capture, manage, store, preserve, and deliver content and documents related to organizational processes. ECM tools and strategies allow the management of an organization's unstructured information, wherever that information exists.

Unstructured information is involved in many fields of today's business life. Documents, e-mails and instant messages are exchanged between colleagues and business partners to share information, coordinate activities or document decisions. This aggregates to a large fraction of the data a company deals with and an effective and efficient management of the content is essential. On the one hand the content has to be managed appropriately and in a comprehensive way (functional requirements). On the other hand, these huge amounts of content must be processed very efficiently. Besides high performance and scalability, cost effectiveness is a key non-functional requirement. By cost effectiveness we mean both reduction in storage and processing cost of steadily growing content collections as well as the administration of the underlying IT infrastructure.

[1] http://www.aiim.org/FAQs/What-Is/What-is-ECM

1.1 Flavors of Content Management

Content management systems manage unstructured data and metadata about them. Pushing further the envelope, with digital rights management (DRM) and knowledge management technology, content along with proper rights became assets and could be traded. With the aid of search and mining algorithms knowledge could be extracted from the content itself thus allowing automatic indexing and classification of content. All this requires a matched infrastructure supporting this kind of business operations.

Actually a wide variety of content management systems already exists. The most widespread ones are:

Web content management: The segment of Web Content Management systems (WCM) is probably the most widespread one. Most books about content management deal with WCM. A WCM system provides a framework for creating, managing, integrating, web-enabling, and delivering unstructured digital content across the enterprise and beyond - to employees, customers and partners. The document life-cycle is elaborated and implies complex workflows. Security plays an important role and performance is paramount for online-business. The size of content objects is usually small, but their number can become large. It's not unusual to find big corporations with hundreds of thousand or even millions of web pages.

Operational content management: Operational Content Management denotes an earlier generation of content management systems deployed primarily by the finance domain, with bank and insurance companies as the early adopters. These systems were used as content repositories for operational content like: bank statements, invoices and recurring print output like monthly reports complementing transaction data. The documents are records of business transactions which are primarily archived and rarely modified. The archive retention period might exceed 10 years and their size reaches the petabyte range.

Rich media: The media market segment is the domain of the rich-media archive systems. With the advances in storage, computing and compression technology, rich-media archives became affordable. At this point the media industry started digitizing their assets and adopting Digital Media Management Systems for their production. With all data digitized, the media business operations became subject to a paradigm shift away from physical assets management and allowing extended re-use of the content and more efficient ways to produce it.

Collaboration + workgroup: The fourth domain is the domain of collaborative and workgroup documents. People are increasingly using unstructured data across

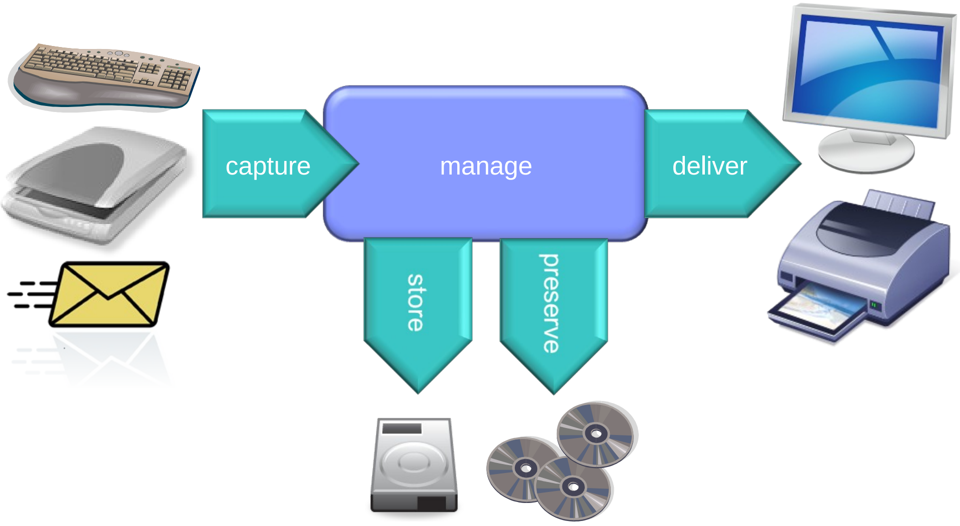


Figure 1.1: Five major components of an ECM system

business environments. In the course of business processes they are leveraging web content, e-mail, spreadsheets, calls, faxes, etc. to perform more efficiently and effectively in their business. The speed of networks, the amount of computing power now available at the desktop and a drop in the cost of storage and processing have aligned so that the use of unstructured data has become not only practical, but also a critical element of conducting e-business. This kind of content management platforms must provide a unified view of multiple repositories so that users can access all relevant content, independent of the application which created it or where it is stored. Business processes in this domain use compound documents and are subject to complex document life-cycles and workflows.

1.2 Components of an ECM system

Based on the definition of ECM one can distinguish five major components of an ECM system shown in figure 1.1 (see also [Kam06]): capture, manage, store, preserve and deliver.

Capture: The capture functions and components are responsible for getting the content into the system. Therefore the content is captured, prepared and processed and finally corresponding entries are created in the system. Due to the wide variety of content in use, the field of capture technologies is also wide. One example are scanners that scan all incoming mails and add them to the content management system. In a preparation step optical character recognition technology can be used to extract text from the documents. In other cases the content is already in a digital form and only has to be captured or crawled, like e.g. files on a computer and e-mails. Further sources of content are web-pages or standalone applications where customers or employees enter information and/or upload office documents. Further input sources are other processes, EDI-messages sent by business partners or records created by business applications.

Manage: While content is in the ECM system it is or can be used. Mainly databases are used to manage the content and to expedite information retrieval. Also access control is important because mostly not everybody is allowed to see all the content, or at least not for free. Often content is not just there, but instead it is used and modified in business processes. Therefore an ECM system has to support the collaborative usage of the content by multiple users, for example through version management, dynamic folders, notifications or workflow management. One further aspect of management is keeping the content as long as necessary, respecting the retention schedules applying to it, and removing it, when it is no longer needed.

Store: The store component is responsible for persistently storing the information and content used by the manage component. Store is typically divided into two functional categories: the catalog and the content repository. The catalog stores primarily metadata like the system state and all the information necessary for searching and retrieval of content. The content repository stores the content in a way that allows retrieving or reconstructing it later.

Preserve: Preserve is also responsible for storage, but especially for the long-term aspects. An important function of this component is migration of the content. When running a content management system over a long time frame, it is very likely that the used storage technology will cease to exist some day. So there will be no additional media or replacement parts. Or, not that critical but even more likely, new technology with lower costs and higher performance will become available. To be able to leverage new technologies, the preserve component must provide mechanisms to migrate large amounts of content from one storage technology to the other.

Deliver: The delivery component is responsible for delivering the content to the user. Typical delivery channels are web-portals as well as mailing printed content or CDs and DVDs. This often involves transforming the content into another format, e.g. the widely used PDF format.

In the previous section we presented several flavors of content management. The respective systems implement the five main components in different ways and to different degrees. Thus the systems as a whole are different, but often there is an overlap at the component level. The developers of the content management applications as well as their users could benefit from common components used by multiple applications.

1.3 Enterprise Content Management as a consolidated Content Management Infrastructure

While the different types of content management systems have differences beyond the application area where they are used, they serve similar needs and provide in essence the same kind of services. The components discussed in the previous section can be found in most of the content management systems. Of course the concrete components can differ from one content management system to the other. The difference is with the domain specific semantics these services offer and with the context in which they are consumed. For example, content creation may relate to scanned documents, a picture or an article. Searching is different in databases, in plain text documents, in audio or videos. Distribution may be done by sending a fax, copying a file or playing an advertising video. Companies are struggling to manage and leverage these disparate systems effectively because they have to be managed separately, and because business content is undergoing an immense growth.

Each of the different content management solutions has to address requirements for mass storage, search and access, personalization, integration with business applications, access and version control, and rapid delivery over the Internet. This commonality suggests that rather than individual point solutions for each type of enterprise content, an ECM framework can - and should - be implemented on top of a set of unified ECM component. By leveraging a common platform and common peripherals the total cost of ownership can be lowered, not only in hardware and software, but also in system administration, training and custom development activities.

Such a framework is depicted in figure 1.2. The layer at the top provides components that support the applications in presenting the content to the users. The applications use a generic ECM interface to call the different functionalities provided by the framework. These functionalities in turn issue create, retrieve, update, delete and search operations on the stored data. Depending on the type of the data different storage systems can be used to actually store the data.

An efficient ECM framework enables applications to cost-effectively leverage the full range of enterprise content. Content can be easily published to Web sites and to portals

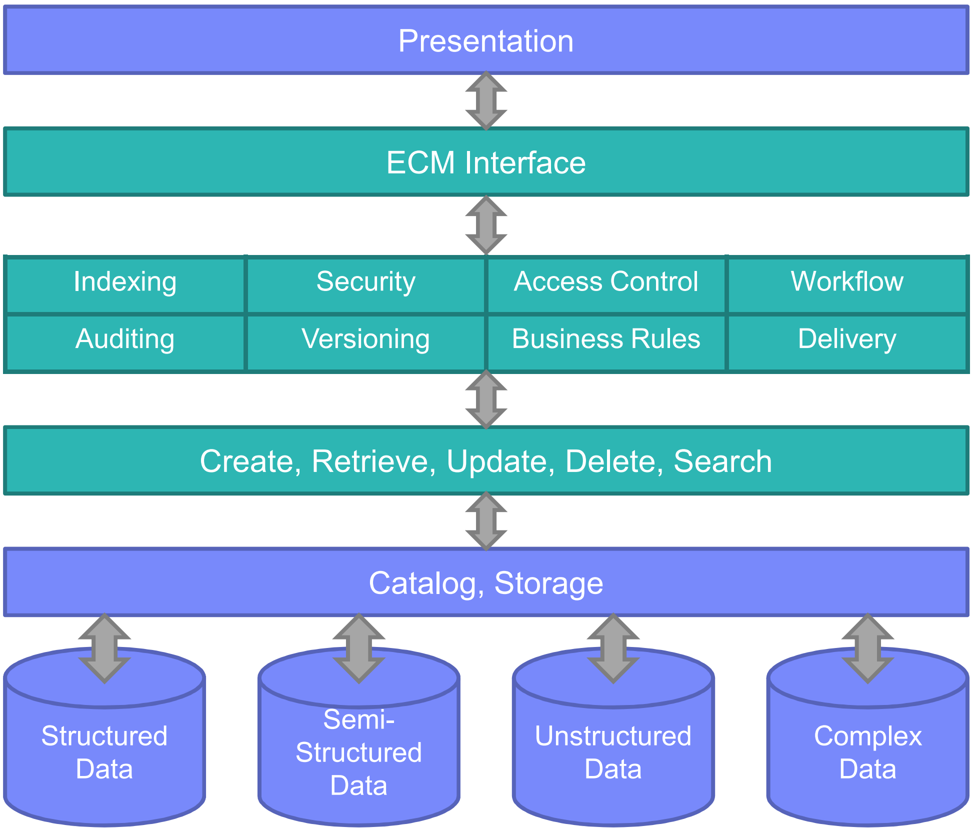


Figure 1.2: Generic ECM system architecture (simplified from [MWM05])

designed to provide secure, personalized delivery to customers, partners and employees across and beyond the enterprise. The ECM infrastructure also allows emerging technologies such as digital rights management or XML-based Web services to be implemented consistently and cost-effectively across different content management applications.

1.4 Contribution of this thesis

In the course of this thesis we look at one specific content management application: E-mail Archiving, Discovery and Management (EADM). The core is to capture e-mails from the e-mail servers, store them as long as necessary to comply with company and legislative regulations, and to make the e-mails accessible to mailbox owners or other persons. The application is described in more detail in chapter 2.

The main focus of the work is on a scalable implementation of the capture process, the storage, and the delivery, including searching for e-mails. Thereby we will design and implement some of the components within a generic ECM framework. The service-based approach allows using them within other applications, too.

The EADM application has some simplifications like other archiving applications or digital libraries. One of these is that most of the data once in the system will never change. This fact can be exploited when implementing the system to simplify the system or to increase the performance. Of course this leads to a reduced general applicability of the components. But there are still multiple applications, like the above mentioned ones, that can benefit from the components.

2 E-mail Archiving

A very prevalent form of content today is e-mails. Not only in personal life, but also in business life a lot of information is exchanged using e-mails. These e-mails are not only plain texts, but may contain a wide variety of attachments and embedded objects. Obviously, there is a huge amount of value (knowledge) contained in e-mail systems. Especially as e-mails are more and more used to execute and close business transactions.

Several studies are available where different institutions measured and estimated the current and future amount of e-mails. The Radicati Group [Rad05] in 2005 counted 133 e-mails per user with an average size of 110 KB. They estimated that the number of e-mails per day will grow to 160 in 2009. In 2006 they conducted a larger study where the executive summary[1] contains some of the findings. They report a growth of the number of e-mails per user and day of 33% between 2005 and 2006. In the same timeframe the average size of the e-mails without attachments grew by 30%, and by 34% for e-mails with attachments. An article in network world[2] from 2000 cites a Ferris Research report according to which the number of e-mails received per user in 2001 grows by 81% to 34 and the average size grows by 192% to 286 KB.

Though the exact numbers may differ, there is a large amount of e-mails sent every day. And all studies agree that this number is growing and will continue to grow in the next years. To handle these masses of e-mails companies already have to provide large amounts of bandwidth and a lot of storage capacity, and the requirements will

[1] http://www.the-infoshop.com/study/rd44619-corp-survey.html

[2] http://www.networkworld.com/archive/2000/85764_01-31-2000.html

still grow in the future. With the amount of e-mails and their size getting larger and larger, the e-mail servers are getting problems and the response times increase. So there are two problems arising from the large amount of e-mails: First the high storage costs, and second the degrading performance of the e-mail servers. Dealing with these problems is one major aspect of mailbox management.

One simple way of mailbox management is enforcing a maximum size of the individual mailboxes. Thus the total storage consumption by the e-mail system is limited and performance problems are also manageable. But this form of mailbox management is not very satisfying for several reasons:

- The user is responsible for staying below the limit. His only options are either permanently deleting some e-mails or exporting them to another location, e.g. his local disk.
- The user has to decide which information is important, and which one can be discarded. This is time consuming and may not always be correct. Especially when hard limits are enforced by locking the mailboxes and the user urgently has to send an e-mail.
- When e-mails are exported to local disks data safety rules are often not fulfilled and the risk of losing valuable information after a disk crash, user error or serious software problem is high.
- Not only the deleted, but also the exported e-mails, are no longer inside a managed environment. Therefore this information is nearly inaccessible within the company.

In summary, simple enforcing mailbox limits come together with a high risk of losing valuable information. Therefore realizing a more sophisticated mailbox management is one of the two major application areas of EADM solutions.

A mailbox management solution offloads the e-mails from the e-mail server and at the same time maintains and preserves the information assets in the e-mails. Therefore the e-mail servers are regularly scanned to identify the e-mails that have to be archived. The offloaded e-mails are stored in a managed, secure and reliable archive. The aims are to comply with laws and regulations and to reduce the costs by either compacting the e-mails and/or using slower and cheaper storage media than for the e-mail servers. E-mail servers typically use high-end and expensive fiber channel disks whereas for an archive SATA disks or even tape libraries can be sufficient. Appropriate tooling makes the archived e-mails still accessible from within the regular e-mail client. For normal e-mail users this facilitates dealing with their e-mails. The advantage for the company is that the e-mails are safe and still accessible when needed. By providing appropriate search functionalities the usability for the user and the company can be enhanced. And finally the archive automatically deletes e-mails when their retention period is over.

E-Discovery is another application area of EADM and is a hot topic for companies because e-mails are more and more used to discuss, prepare, and communicate business decisions. Therefore at least a subset of the e-mails has to be treated as business records.

The financial scandals about the US enterprises Enron and WorldCom (now MCI) have shown that the information in old e-mails can become very important for different entities one day. Therefore many laws and regulations were put into place enforcing regulatory compliance and archiving of electronic documents like e-mails [Thi04, U.S06]. Additionally, existing laws and regulations demanding safekeeping of documents are nowadays also applied to digital documents and e-mail communications. One example is rule 17a-4 of the US Securities and Exchange Act which requires that a specific set of persons "shall preserve for a period of not less than 6 years, the first 2 years in an easily accessible place, all records required to be made pursuant to [a set of rules]"[3]. Another sample law in the area of financial reporting is the "Public Company Accounting Reform and Investor Protection Act of 2002" better known as "Sarbanes-Oxley Act" (SOX). SOX is a United States federal law explicitly enforcing comprehensiveness of record management (i.e. paper, electronic, transactional communications, which includes e-mails, instant messages, and spreadsheets that are used to analyze financial results) and to maintain records for certain periods of time.

Besides the necessity to comply with legal requirements related to certain business areas, e-mail archiving is nowadays a general obligation for executive care that is indispensable, since more and more companies are forced by litigation to (re)produce old e-mails. Large companies can be faced with several of these requests per day. Without adequate procedures fulfilling such requests is a very time-consuming and expensive task. Additionally, it is error-prone, which in turn could lead to (further) problems at court. Some cases became known, where institutions were forced to reproduce e-mails from backup tapes causing costs of several hundred thousand dollars [Plo04]. Others paid a large amount of money to settle the case out of court because the reconstruction was too expensive or even impossible [SE05].

Nowadays the EADM system that supports e-discovery is often a separate system which is only used for compliance checks and to fulfill court orders. Companies are implementing such systems only because they have to. In some domains the executive management of the company is personally responsible for the correctness of this archive.

[3] http://www.law.uc.edu/CCL/34ActRls/rule17a-4.html

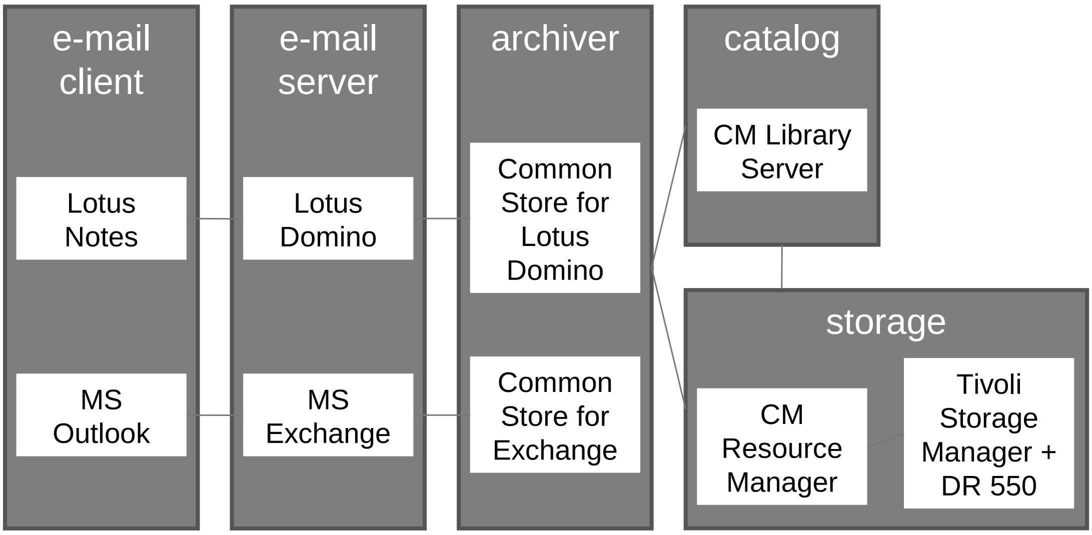


Figure 2.1: Typical implementation of an EADM solution

Furthermore in some domains incoming and outgoing e-mails have to be checked for compliance with a given set of rules to prevent illegal or inappropriate communication. Sample topics for e-mail filtering are: data confidentiality, insider information, (sexual) harassment, and protection of data privacy. Companies were sued for such e-mails sent by their employees. To reduce this risk, suspicious e-mails are presented to a supervisor before they are delivered. Of course, filtering cannot completely prevent non-compliant communication. There are always ways to hide information. However, accidental communication, by e.g. picking the wrong receiver, can be prevented to some extent. Having reasonable measures in place at least lowers the risk for the company.

2.1 E-mail Archiving, Discovery and Management

Figure 2.1 shows the implementation of an EADM solution loosely based on the reference implementation of an e-mail archiving and retention solution presented in [LCHC^{+}07]. On the left hand side there are the e-mail clients like Lotus Notes or Microsoft Outlook that are used to compose, send, receive and manage the e-mails. Second to the left there is an e-mail server like Lotus Domino or Microsoft Exchange that stores the e-mails of its users and routes incoming and outgoing e-mails. The e-mail clients and e-mail servers are established off-the-shelf products with plug-ins or extensions to manage archived e-mails.

The only component in this figure that is specific to e-mail archiving is the archiver itself. It is bridging the gap between the e-mail server that manages the online e-mails, and the content management system that manages the archived e-mails. It continuously or periodically monitors the e-mail server to locate e-mails that are eligible for being archived. These e-mails are retrieved from the e-mail server, processed in several ways, and are then ingested into the content management system. Finally, depending on the configuration, the e-mails are either completely removed from the e-mail server, or are replaced by a much smaller stub containing a reference to the archived e-mail.

The content management system is depicted in the figure as two components: the catalog and the repository. The catalog manages the system and the documents. To this end all relevant information to manage, search and retrieve e-mails is stored in a database and/or a search engine. The catalog is a central component that is involved in all operations by the users or the archiver. The repository is responsible for storing the archived e-mails. It is basically like a file system and stores plain files, but can provide additional functionalities like e.g. hierarchical storage management or replication.

The repository is backed by storage management and storage devices. Typical storage devices are disks, tapes and CDs or DVDs. In many scenarios the storage system must ensure that the archived documents are not tampered with. One way is to store the documents on media that can be written only once by their very nature, e.g. CDs or DVDs. But these are not very fast, and the media has to be destroyed and disposed when it is no longer needed. Therefore several vendors provide integrated disk systems where the software guarantees a write-once behavior.

Not shown in the picture are optional components that can be added to the content management system. One example is a component specific for e-Discovery. They are designed to search in large collections and to work with and export large result sets. Another example is the records management, where the archived e-mails can be marked as business records and are henceforth managed following the rules for the particular type of record.

2.2 Functional Requirements

In contrast to content management in general it is a special property of EADM that e-mails are never updated. Once they are sent they cannot be modified anymore. In relation to the amount of data to be considered, most e-mails will never be retrieved but are managed until the end of their retention period and then they are disposed. Therefore, the major concern of EADM systems design must be the optimization of the ingest process. Classification and filtering of e-mails as well as persistence mechanisms with different properties, e.g. for records management, are other required functions.

In the following we describe the primary EADM use cases. Of course there are many more, but here the focus is on those relevant for the message of this work, especially ingestion of the inbound message flow from the various e-mail sources into the archive.

Use Case 1 - Ingestion of e-mails: The EADM system connects to an e-mail server and retrieves the e-mails. There are two different approaches to capture the e-mails: 1) A crawler regularly scans all mailboxes and archives the e-mails within these mailboxes that meet specific criteria defined by rules. Examples for such rules are to select all the e-mails that are older than 3 month, or to select all e-mails with a size larger than 1 megabyte. Archived e-mails are either completely removed from the mailbox, or are replaced by small stubs with references to the copies in the archive. It is possible that the EADM system misses e-mails that are stored and deleted between two consecutive runs of the crawler. When compliance and e-discovery are important, this is not acceptable. 2) To capture all e-mails processed by the e-mail server, Lotus Domino and Microsoft Exchange provide journaling mailboxes. The e-mail server stores a separate copy of each incoming and outgoing e-mail in this journal. The EADM system then uses an adapted crawler to retrieve the e-mails from these journal mailboxes.

Use Case 2 - Search: A regular user wants to retrieve an old e-mail. He remembers some part of it like the sender/receiver of the e-mail or some words/parts of the subject or body of the e-mail. For this use case the EADM system has to provide means to search for archived e-mails using the following criteria:

- a time frame when the e-mail was sent
- sender and recipients of the e-mail
- a set of words in the subject
- a set of words in the body
- a set of words in the attachments

The hits are presented to the user in a table with several fields of the e-mail.

Use Case 3 - Retrieve: The user has identified an archived e-mail he wants to look at by either executing a search or opening a stub of an archived e-mail in a mailbox. The search result and the stub both contain information that is used by the EADM system to retrieve and reconstruct the e-mail. The e-mail can then be directly displayed to the user or it can be re-imported into a mailbox.

Use Case 4 - Discovery: A privileged user like e.g. a compliance officer looks for e-mails that violate rules or that are relevant to a case. Therefore he might issue a search request for all e-mails sent and/or received by a given set of users within a certain period of time and containing a specific company name. These search requests encompass all mailboxes or large subsets thereof and thus can produce a large number of hits. To further work with the hits they can be arranged into folders. The set of e-mails in a folder can be further refined or they can be exported for being further examined by e.g. lawyers. As long as an e-mail is in such a folder it is also put on legal hold in order to prevent its deletion.

Use Case 5 - Expiration: The last use case covers the removal of e-mails. The archive regularly deletes all the e-mails that have reached the end of their retention time, and are not on a legal hold.

2.3 Non-functional requirements

The EADM system may become a very important part within an enterprise. Therefore it has to face several challenging non-functional requirements.

Important non-functional requirements for an EADM system are reliability, authenticity and security. These characteristics are mandatory for an archive in a real-world environment. Without, the results of the archive are not usable in legal cases. And a regular user would search for trustable alternatives.

The performance is also an important requirement. On the one side ingesting new e-mails into the archive has to keep pace with the incoming e-mails. Yet the daily time window available for archiving is often restricted to also permit administrative tasks. Furthermore during the deployment of the system a large backlog of old e-mails has to be processed. On the other side the response times and the throughput for the interactive users have to be adequate.

One major challenge regarding the development of an appropriate EADM system is to support the required e-mail processing, archiving and search functions in an efficient and scalable way. The data volumes and the load on the system are seldom constant. There are short-term fluctuations within a day or a week, and there are long-term changes. Regarding retention periods of seven years or longer and the size of the archive, it will be very difficult to replace an EADM system when it no longer fits the requirements.

Regarding the availability of the system the requirements towards an EADM system are not as high as for an OLTP system. But still there are regulations that require an online access to the system. Thus availability is not a top priority, but it is still important.

Further important requirements are affordability and cost-effectiveness, especially in the small and medium businesses segment. The system capability to dynamically adapt to the current situation by integrating additional resources or releasing unused resources is worthwhile.

2.4 Related work

As e-mail archiving is an important issue, there are many solutions available on the market [DC07]. Sadly enough, the public information is often limited to some marketing brochures. When comparing the EADM processing pipe to existing ECM capabilities, one can clearly identify common tasks: indexing and persistent store, as well as search and retrieval are common. Thus several EADM systems are backed by an ECM system. Other CM functionalities like workflow are obsolete for EADM. Yet other components like the crawler are specific to EADM. Some existing solutions are presented in the remainder of this section.

The rough architecture of IBM CommonStore [ZBB+06] was already shown in figure 2.1. CommonStore allows setting up multiple instances that are all archiving into one IBM DB2 Content Manager [Che03]. So crawling can be scaled out, and by adding more repositories it is possible to scale-out the storage. But the central catalog can become a bottleneck. For e-discovery an add-on called "eDiscovery Manager" exists. For compliance supervision IBM is partnering with other companies. Before the acquisition by CA Inc. the partner was iLumin. This solution is mentioned here because the available documentation [Yun05, ZFH+06] gives a good overview of this solution.

Mimosa NearPoint for Microsoft Exchange Server [Spu07] is capturing the e-mails by reading the log files of the e-mail server. From these logs it builds and manages its own copies of the exchange databases in order to reduce the load on the e-mail server. The process called "smart extraction" performed in one of two grid architectures extracts information from these databases and processes it further. The extracted information is stored in a common back-end storage and a common Microsoft SQL database.

The AXS-One Compliance Platform is integrated in the Sun Compliance and Content Management System [Mic07]. The system operates in either compliance or operational mode. There is one archive engine and there can be multiple archivers, but the available description is very high-level.

The Symantec Enterprise Vault [Sym06, ZGK+06] allows archiving content from different sources, and e-mail servers are one of them. The system is made up of different services, with the most important ones being the storage service responsible for compressing, storing and converting the content to HTML, and the indexing service providing search capabilities using AltaVista search technology. It is possible to instantiate these services on different nodes, whereas it is recommended to collocate corresponding mailbox tasks, storage services and indexing services. The system can be extended by different accelerators, e.g. a compliance accelerator to regularly review the e-mails, and a discovery accelerator for enterprise-wide search.

A few research projects also looked at aspects of e-mail archiving. The "Texas Email Repository Model" [GSGG02] focuses on the aspects of long-term preservation for a statewide e-mail archive. A project within IBM research [MHW06] considered a very stringent fraud model. They make sure that nobody in the enterprise, no matter how many privileges he has, can manipulate the data or metadata of a record after it was archived.

2.5 Summary

In this chapter we gave a short introduction into the area of e-mail archiving. We started with some numbers that show the dimensions of e-mail archiving, especially the large and growing amount of e-mails sent over the Internet. We then mentioned some of the technical, legislative and regulative reasons why archiving these e-mails is so important for enterprises.

With one example we showed how a typical implementation of an EADM system may look like. We described how the different components of the EADM implementation work together to provide their service. After that we got into more detail about the functional and non-functional requirements such a system has to fulfill.

Finally we looked at different existing EADM solutions and research projects in the area of e-mail archiving.

3
Virtualization and Scalability Issues

As mentioned in section 2, EADM systems have to deal with large amounts of data. The original documents are the largest fraction, but the metadata and indexes can become large, too. To store and manage this data, an adequate sized system is required. When initially sizing an ECM system several parameters like the expected data volumes, the workload induced by processing the documents and fulfilling user requests, the dynamics of the workload with peaks, and the costs have to be taken into account. With the above mentioned parameters on the one side and the required performance characteristics of the system on the other side, one can determine an initial sizing of the system.

But things are changing: the average size of the document is growing, the workload increases, the number of users is increasing e.g. due to the popularity of the system or to mergers with other companies, ... Thus the load on the EADM system almost certainly will change gradually. On the other hand cost effectiveness prohibits oversizing the system. It is therefore inevitable for a system to be expandable on demand. Otherwise huge problems will arise when after several years large amounts of valuable information are locked into a system that has reached its capacity boundaries.

The two important features for the EADM system to this end are scalability and adaptability. Scalability roughly means that the performance of the system increases with more available resources. Ideally, twice the amount of storage allows doubling the number of documents, and doubling the number of processors increases the number of requests that can be processed by a factor of two.

Being able to exploit the available resources efficiently is important. Yet scalability only looks at different static configurations of the system. A system can be scalable even when adding new resources requires making a backup, shutting down the system, adding the new resources, bringing up the system again and restoring the backup. With the expected large data volumes this definitely does not resolve the problem. The system must be able to seamlessly adjust to the new situation by incrementally reorganizing the state.

As a consequence it is not very realistic to set up a static system and to suppose that this will do the job for all times. To overcome this problem a virtual system is introduced which subsumes and hides the currently assigned physical resources. The virtualization will decouple the users of a service from the actual resources that provide the service. Under this hood the system can grow and shrink.

In this chapter we first look at scalability. We discuss different ways to scale a system and different system architectures. The second part of the chapter covers different forms of virtualization.

3.1 Scalability

According to Neuman [Neu94], a distributed system scales "if it can handle the addition of users and resources without suffering a noticeable loss of performance or increase in administrative complexity. Scale has three components: the number of users and objects that are part of the system, the distance of the farthest nodes in the system, and the number of organizations that exert administrative control over piece of the system." This definition is not very precise, but it mentions some important aspects of scalability. Especially that scalability is not only a performance issue, but also an administrative issue.

There are several approaches to more formally define the term "scalability" (see e.g. [Hil90, JW00, DRW07]). We want to roughly mention [JW00]. A system is said to scale if it can be efficiently utilized in a wide range of configurations. A configuration is an instance of the system using a concrete set of resources. It provides the required functionality with a measurable quality of service and induces different costs. The approach computes an efficiency of a configuration by correlating the quality of the service the configuration provides with the amount of resources that are necessary for the configuration. To determine the scalability between two configurations the ratio of their efficiencies is calculated. A ratio of one means that the performance of the system growth proportionally to the costs between the two configurations. The system is said to scale linearly between the two configurations. When the ratio is larger than one the second configuration is even more efficient. This can happen e.g. when the resources needed for the smaller configuration are still sufficient for the larger configuration with more users and hence for a higher throughput. When the ratio is smaller than one, then the second configuration is less efficient than the first one. For example when the costs for the second configuration are twice as high, and the performance increases by only 80%, then the ratio is 0.9, assuming the quality of service is the same. This is not optimal, but to some extent often still acceptable.

Amdahl uncovered in [Amd67] one inhibitor for parallelization and scalability: the sequential fraction of an algorithm. This sequential fraction is often due to accessing a shared resource or using a synchronization mechanism to ensure a globally consistent state. The problem is, that no matter how many resources are used, the time used within the sequential fraction of the algorithm cannot be reduced. With unlimited resources the time used for the parallelizable fraction approaches zero, but the sequential fraction stays the same. So e.g. with a sequential fraction of 10% the maximum achievable speedup is 10.

For a system that has to be scalable this means, that the sequential fraction of all algorithms has to be low. Especially a centralized coordination should be avoided as far as possible, or at least the interaction with it should be minimal. Of course, as always, there is a trade-off: in a small system a central coordination can be faster, whereas in a large system with hundreds or thousands of participants a distributed approach is advantageous.

Besides the technical aspects of scalability another important aspect is administrative scalability. For a production system it is not enough that it is theoretically possible to set up the system and to operate it. The necessary administrative effort also has to be taken into account when evaluating the system. For example in the formal model mentioned above, the administrative costs can be incorporated into the costs of the configuration. The administrative scalability can in fact be an inhibitor for the scalability of the system. Without appropriate tools and procedures in place, the administration of large and dynamically adjustable systems can become a struggle.

3.1.1 Scale-up vs. scale-out

To realize a scalable system there are two basic approaches as shown in figure 3.1:

Scale-up / scale vertically: In a scale-up approach either additional resources are added to a single system or the application is completely moved to a larger system. Typical examples are to assign or add more CPUs or more memory to a computer. The additional resources can then be exploited by e.g. starting more threads to handle the incoming requests. In times with weaker utilization, excess resources can be assigned to other tasks using hardware partitioning or virtu-

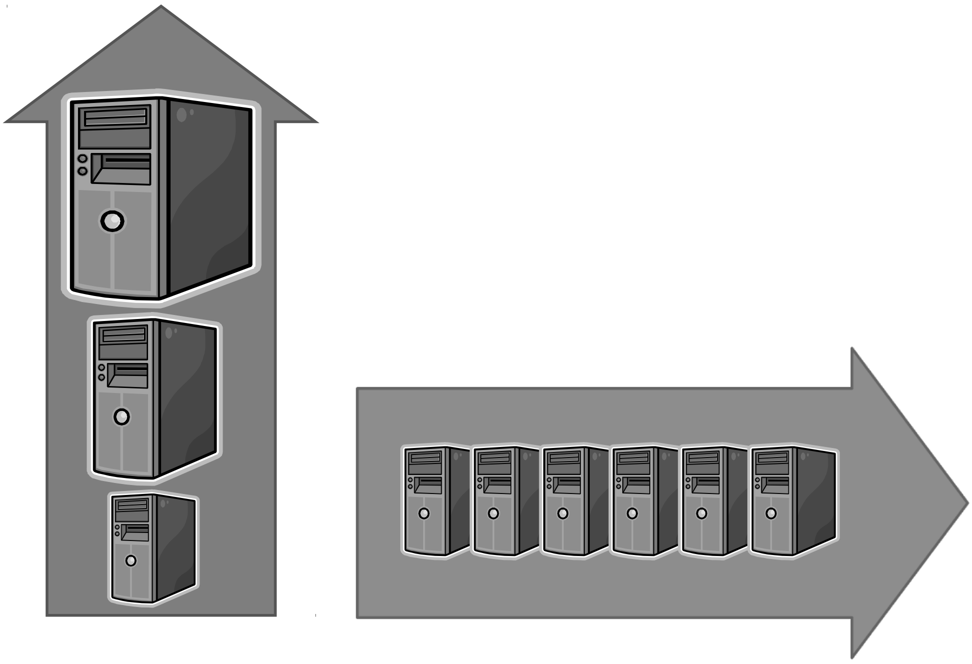


Figure 3.1: Scale-up vs. scale-out approach

alization techniques. Scale-out breaks down when the single system reaches its maximum stage of expansion. Moving to a completely new system will incur large costs and very likely a service interruption. Therefore right from the start a system has to be purchased that can grow up to the maximum envisioned capacity. E.g. there must be enough spare CPU sockets to later increase the number of CPUs when needed.

Scale-out / scale horizontally: In a scale-out approach the capacity of the system is increased by e.g. adding more whole computers. Typically inexpensive, commercial off-the-shelf (COTS) whole computers are used. Thus it is possible to start with a small configuration and one does not have to pay the surcharge for the premium technology that might be required some day. Yet, more machines are more complex to manage than a single machine. Without appropriate tools this will become a nightmare for the administrators.

In a scale-out approach the fact whether the resource to scale is stateless or not plays an important role. With stateless resources like most web application servers the main challenge is to hide the internal complexity from the outside and to make them usable like one resource. With stateful resources, and in particular databases, the state introduces further challenges: 1) the state has to be synchronized, and 2) the state has to be stored somewhere. For large data sets it can be quite resource intensive and time consuming to reorganize after a structural change and it thus has to be considered in more detail upfront.

In an attempt to compare scale-up and scale-out several IBM laboratories conducted a study to evaluate the scalability of different architectures for commercial workloads. One of their workloads was from the area of unstructured information management, where they built a search application based on the two Apache projects Nutch and Lucene [MMDS$^+$07, MMSW07]. This workload was evaluated on two different systems: The scale-up system was an IBM POWER5 p5 575 with 16 processors, and the comparably priced scale-out system consisted of IBM BladeCenters with different kinds of blades. Their final conclusions were: 1) The scale-out system has a clear performance and price/performance advantage. 2) Even on the scale-up system it was more effective to implement the application in a scale-out fashion with multiple communicating processes than using only one large process. 3) The scale-up system has the advantage of simpler system management and thus lower management costs.

3.1.2 Limitations for scale-up

In a pure scale-up approach more CPUs or more memory are added to the system, which are then used by the application. Depending on the physical resources it is possible to assign large amounts of resources to one application. In this section we mention two reasons that can limit the success of this approach.

First there are limitations mostly enforced by the software. They occur in many different forms. Some of them are:

- Maximum amount of memory that can be used by one application, especially for 32-bit applications.
- Maximum size of files in the file system. This may restrict the size of persistent data structures.
- Databases have limits for the number of rows per table, the size of a row, the size of a table space, …
- The size of identifiers. For example Apache Lucene, the full-text search engine library we are using in later sections, currently uses a 32-bit integer to identify the documents. This limits the maximum number of documents in one index to around four billion. With an expected number of 25 million e-mails per day this lasts for only 172 days.

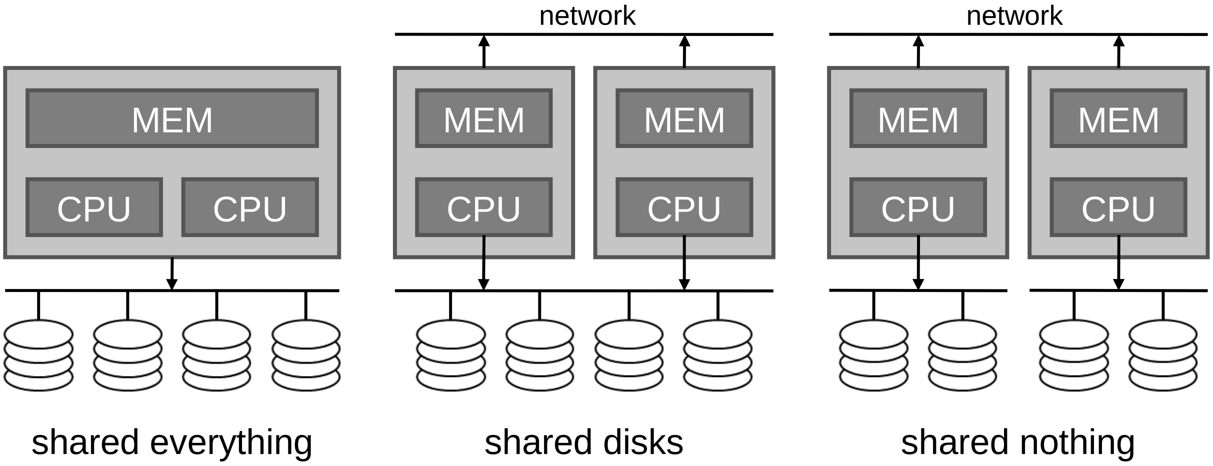


Figure 3.2: Multiprocessor system architectures

Another reason that can limit the scalability of a system is a consequence of the simple programming model that can be followed with a scale-up system. As it is possible to build one big application that has access to all the resources and all the data, it is not necessary to separate the application into multiple parts and to think about the communication between different components of the application. But this can lead to a contention between the shared resources. Therefore, even on a shared-everything system it can be beneficial to separate the application into several components.

Due to the size limitations it is absolutely necessary to break up the index into multiple segments. Furthermore having the index separated appropriately can be also beneficial for the processing performance. Of course the different functionalities of the EADM system have their own characteristics that have to be taken into account.

3.2 Multiprocessor system architectures

Different system architectures have been proposed to realize scale-up or scale-out. One common way to classify these architectures is by the resources, disk or memory, that are used in common, and where all CPUs have shared access to. The three main architectures are shown in figure 3.2:

Shared everything: In these systems all the processors access the same memory and the same disks. As all the processors have equal access, the distribution of data and processes is of lower importance. The communication and synchronization occurs over the bus between processor and memory which has a relatively high bandwidth. Yet this bus has to provide a high bandwidth already for a single processor and the bandwidth has to increase with each additional processor in order to avoid a bottleneck. The high bandwidth also limits the possible physical extension of the system. Typical representatives of this architecture are symmetrical multi processors (SMP). With the strong rise of multi-core processors most systems nowadays are at least partially SMP systems. Due to the mentioned bandwidth requirements, existing SMP systems rarely have more than 16 processors. Systems with a non-uniform memory architecture (NUMA) are one approach to alleviate this problem. In principle each processor can still access the whole memory, but each part of the memory is associated to one processor. Accessing the memory attached to another processor is remarkably slower. Thus with a NUMA system the assignment of processes to different processors has an impact on the performance.

Shared disk: With this approach the processors use the same disks to store information persistently. Yet every processor has its own memory to buffer the information from the shared disks and to work with it. To synchronize the memory between the processors the data has to be written to the disk or it has to be exchanged using the network. As the disks have to be accessed from multiple machines, they are provided by a storage area network (SAN) or a network attached storage (NAS) system. Such a system can be easily extended with more machines, as long as the disk system and its connection is fast enough.

Shared nothing: In this approach the disks are also assigned to a processor. The only data exchange between different processors occurs through the network. Thus processes must (at least partially) be executed by those processors that own the required data.

All these systems have shared communication media (network, disks, memory) that may become a bottleneck. The time fraction a process occupies the shared resource can be seen as one sequential fraction of the algorithm. Thus, according to Amdahl's law, a limit exists beyond that adding more processes or processors will not increase the performance of the system any more.

The capabilities provided by the hardware are one issue. Whether they are exploited by an application is another. For example the limitations mentioned in section 3.1.2 may lead to a shared-everything system that intentionally does not fully exploit the shared memory. The separation into multiple components that do not use the shared memory can be beneficial regarding performance, availability and architectural aspects. Similarly, it can be beneficial to use a shared disk by only one processor.

For a more in-depth discussion of multiprocessor system architectures see for example [Pfi98].

The prime architecture we favor for the EADM system is a shared-nothing architecture. This architecture can be implemented using commodity hardware which reduces the costs of the system. It must also be possible to increase the capacity of the system even several years later. It is more appealing to add complete nodes with the off the shelf technology than to buy costly extensions for an old system.

3.3 Virtualization

Virtualization is a hot topic in the computer industry. There are many different kinds of virtualization. All of them have in common, that the direct access to some resource is cut off. Instead the consumer accesses an intermediary, a virtual resource. This virtual resource then forwards the requests to an actual, physical resource.

The advantage is, that the user of a resource is decoupled from the resource and is no longer statically bound to it. This alleviates the replacement of the resource by another when e.g. the initial resource has failed or is no longer powerful enough to deliver its service with the required quality of service.

Of course the introduction of an intermediary may add some overhead that can reduce the performance of the virtual resource compared to using the physical resource directly. Thus an important requirement towards the virtualization technology is to minimize the overhead so that the benefits from the additional flexibility are higher than the loss in performance.

As already mentioned there are many different forms of virtualization. First we look at infrastructure virtualization, where the software uses virtual resources that can be mapped more flexible to real resources. The second form is database virtualization, where distribution of the data is hidden inside the database. And finally we look at service virtualization, which allows orchestrating and deploying a software system more flexible.

3.3.1 Infrastructure virtualization

One kind of infrastructure virtualization that is very interesting for data center operators is the server virtualization. In 2006 Metrics Based Assessments reported[1] that the utilization of 79.1% of the UNIX production processors and 94.3% of the windows production processors is below 20%. There can be many reasons for this, but an important one is the static configuration and deployment of the applications. A given server is acquired, installed and configured for a specific application. If dynamics are

[1] http://www.metricsbasedassessments.com/featured_metrics/2006-06%20MOM%20Server%20Utilization.htm

not taken into account from the beginning, and an appropriate environment is not set up, the static configuration will manifest itself. After some time of use it is too risky to change this system. For each and every application resources have to be provided that can withstand the expected peak demand. As a consequence a large fraction of these resources are idle most of the time.

Using server virtualization applications are not directly installed on a physical machine. Instead, they run inside a virtual machine. Examples of such machines are VMware[2], Microsoft Virtual Server[3] and Xen[4]. These virtual machines can be started on the appropriate physical machine. In some cases even a live migration of a running virtual machine from one physical machine to another is possible. In periods with low demand multiple virtual machines can be hosted on one physical machine. The superfluous machines can then be assigned to other tasks, or they can be shut down until they are needed again. Thus server virtualization is an important technology for an efficient utilization of data centers and for a green IT.

Another form of infrastructure virtualization is storage virtualization. With storage virtualization an additional controller is introduced between the servers and the disk subsystems. This controller provides virtual disks for the servers and maps them to the real storage. By changing this mapping the capacity and the performance of the virtual disks can be adjusted. Storage virtualization also provides location independence, which is also important for server virtualization: No matter on which physical machine a virtual machine is started, it needs access to its data.

Nowadays it is no longer necessary to host these services oneself. With Cloud-computing [AFG$^+$09, FZRL08] it is possible to buy such virtual resources from a third party. They can be bought via the web with just a credit card, and the time until they are available is within a few minutes. One example offering are the Amazon Web Services [5]

With the EADM system we are not explicitly focusing on a virtualized infrastructure. We are more concerned about using the available resources, and not so much about where they are coming from. But for large installations, or when offering EADM to customers as a service, a virtualized infrastructure will simplify the necessary administrative effort.

[2] http://www.vmware.com/

[3] http://www.microsoft.com/windowsserversystem/virtualserver/

[4] http://www.xen.org/

[5] http://aws.amazon.com/

3.3.2 Virtualization using distributed database technology

On a higher level in the software stack it is possible to e.g. virtualize the database. Distributed databases allow using the resources of multiple nodes for one database (see e.g. [Dad96] for an introduction). This enables a scalable database system as additional resources can be added when needed. The distribution of the database is completely transparent to the client applications, which can still use the database just like a single-node database.

Many database vendors provide extensions that allow defining a database instance that spans multiple nodes. For the IBM DB2 for example this is provided as part of the Enterprise Server Edition and the feature is called "Database Partitioning Feature" (formerly called Enterprise Extended Edition). It supports partitioning a database within one server or over multiple nodes [BFM+03]. On a table-by-table basis one can decide on which node to store the table, to replicate the table on a set of nodes, or to partition the table and to assign the partitions to different nodes.

Additional nodes can be added to the database and superfluous nodes can be removed from the database on demand. If a table is explicitly placed on a node, the administrator will have to adapt the allocation of the table. Replicated and partitioned tables can be reorganized using the "reorg" command. This reorganization requires a service interruption.

Such a distributed database still has limitations, but provides more options to adapt the database to the current requirements. At the same time it provides the same security and consistency properties like the single-node database. In section 5.2.7 we report some experiences with a distributed database underneath an ECM system.

3.3.3 Service virtualization

At a more abstract level a service-oriented architecture supports the virtualization of different components and functionalities at all levels of the software stack. The concrete implementations are wrapped by a service that often has a more general definition. This has several advantages:

1. The services can be used by different applications,

2. The services can be deployed separately to satisfy performance requirements, and

3. The services can be replaced by a different implementation.

[WCL$^+$05] define a service as follows:

> A service is available at a particular endpoint in the network, and it receives and sends messages and exhibits behavior according to its specification. The service has specific functionality and is deployed with appropriate quality of service. [...] Interfaces and policies describe the terms and conditions that govern the use of the service. These are published so that potential users of the service can discover and be given all the information they need to bind (perhaps dynamically) to that service.

An application may use many different services. An architecture that supports building an application in this way is the service-oriented architecture (SOA). A middleware is used to interconnect the different services. This middleware is sometimes called an enterprise service bus.

Within a SOA performance and scalability are addressed via an intelligent workload management system, using schemes from system automation and dynamic load balancing. In tandem with high availability the requirements can be accomplished by:

- Automatic and dynamic system (re-)configuration
- Automatic and dynamic routing of application requests to the services
- Growing and shrinking the environment as needed for a more efficient resource usage, which helps satisfy the affordability.

Web Services [WCL$^+$05] are a very well-known kind of services. Web Services are defined by a set of standards that define aspects of the implementation, the description of the service, the discovery of the service and the consummation of the service over the Internet. Many additional standards add features like security, the representation of state and transactions.

Grid Computing comes from the area of supercomputing. Initially the focus was on virtualizing compute resources and access to data in the form of files using custom protocols. With the Open Grid Services Architecture (OGSA, [FK03]) and the Open Grid Services Infrastructure (OGSI) they turned to used Web Service standards. OGSI is based on Web services and primarily enhances them to support long-living, persistent services that are often needed within distributed applications. To this end it provides mechanisms for creating, naming, managing the lifetime, monitoring, and grouping of services, and for information exchange. An open source implementation of OGSI is the Globus Toolkit version 3[6].

[6] http://www.globus.org

Based on OGSA, the Database Access and Integration Services Working Group of the Global Grid Forum[7] develops standards to access databases. The OGSA-DAI project[8], which is part of the UK e-Science project, developed an implementation that supports the access to existing relational and XML databases.

As part of the eDiaMoND project[9] the IBM DB2 Content Manager was integrated into OGSA-DAI. Yet they do not fully exploit the IBM DB2 Content Manager. They use it to store and retrieve large files, in the concrete case images of their patients. But they do not really use the metadata management functionalities. Instead they store their metadata in regular databases and use the IBM DB2 Information Integrator to integrate the information from the different sources.

Within the Grid community a system that is widely used to manage files is the Storage Resource Broker[10] (SRB) of the San Diego Supercomputer Center. It allows to access many different storage resources and manages metadata in a centralized Metadata Catalog (MCAT). With zoneSRB metadata and data from different SRBs can be shared and replicated between the local MCATs based on policies [RWMS04]. Yet this approach only provides federated access to the SRBs, and does not organize the data.

3.4 Summary

In this chapter we first defined the term "scalability". After this definition we discussed the two approaches to scale a system: scale-up and scale-out. For the scale-up approach we listed some of the reasons that can limit the scalability of a scale-up approach. In the following section we introduced the three basic system architectures that can be used to realize different forms of scalable systems.

Finally we looked at different forms of virtualization. Virtualization introduces a layer of abstraction on top of concrete components or services that allows decouple them from their users. This decoupling is a valuable feature when building a dynamic and scalable system.

[7] http://www.gridforum.org/gf/group_info/view.php?group=dais-wg

[8] http://www.ogsadai.org.uk/

[9] http://www.ediamond.ox.ac.uk

[10] http://www.sdsc.edu/srb

4 Architecture of the EADM System

In section 2 the application domain EADM was already introduced. Based on the requirements set, we present a system architecture for an EADM system in two steps: The first step starts in section 4.1 and presents a single-node architecture that fulfills the functional requirements. We also demonstrate how the major use cases are realized on this architecture. In the second step starting in section 4.4 we extend the architecture to improve support of the non-functional requirements performance and scalability. To achieve this, virtualization is introduced to deploy the architecture in a distributed system.

4.1 Single-node architecture

Figure 4.1 shows the single-node architecture of the EADM system prototype that we first presented in [WKM$^+$08b]. The system is separated into three layers: the data layer, the repository layer and the application logic layer.

The data layer is responsible for storing the data of the EADM system. This data encompasses the archived artifacts, but also metadata about the artifacts necessary for example for the management of their lifecycle. Additional metadata is required to support an efficient search and retrieval of information from the archive.

The types of the data in an e-mail archive range from structured data over semi-structured data to unstructured data. The different types of data are also used differently and there is not one technology that ideally manages all the data. Instead of

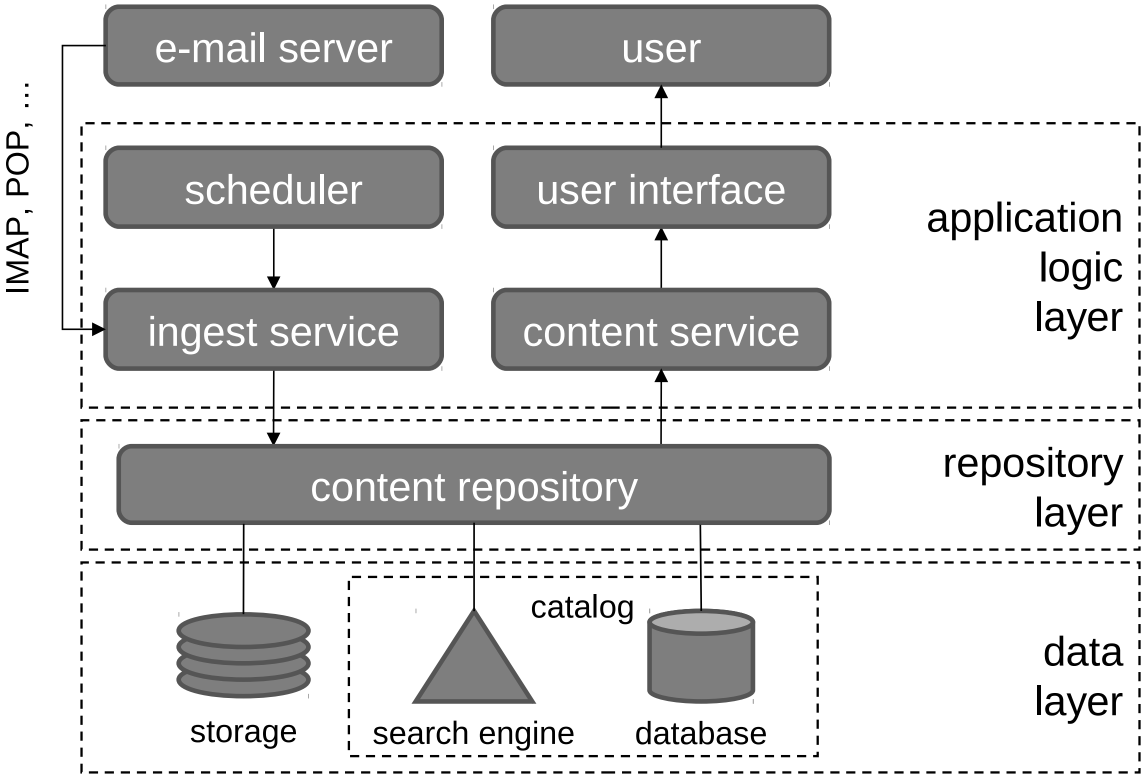


Figure 4.1: Architecture of the EADM system

forcing all the data into one system, we use different systems for different parts of the data. The storage that is used to store the e-mails in their original, unstructured form could simply be a file system. The catalog that manages the metadata comprises of two parts: The search engine is used for the semi-structured textual information from the e-mail, and the database is used for the structured part of the data. A more detailed description of how the data is separated is given in section 5.1.

Above the data layer is the repository layer. Using different systems in the data layer to exploit their particular advantages is one thing. But they also have to provide a service together. This is the task of the repository layer. It has to combine the different systems in the data layer into one logical system. When archiving an e-mail, the repository layer knows to which systems in the data layer it has to forward the e-mail. And when a user searches for e-mails or wants to retrieve one from the archive, the repository layer forwards these requests to the appropriate system.

The application logic layer consists of those components that are specific for e-mail archiving. They are bridging the gap between the world outside the EADM system and the repository. On the left side of figure 4.1 are components responsible for ingesting new documents into the system. The scheduler knows when the different mailboxes have to be processed and with which parameters. It therefore maintains a work queue with jobs that have to be performed. These jobs are processed by the ingest service. In short the ingest service retrieves the e-mail from an e-mail server and stores them in the repository. It is discussed more thoroughly in the following section.

On the right hand side of the application logic layer in figure 4.1 are components retrieving information from the system. The content service retrieves the information from the repository layer and prepares it for the user interface. The user interface is a web application that provides mechanisms to search for e-mails and to display them in a browser. It also provides mechanisms for users like a compliance officer to organize large subsets of the e-mails into case folders.

4.2 Ingest service

The ingest service is the component that performs the ingestion process of new e-mails into the EADM system. It retrieves the e-mails from an e-mail server and archives them. Existing EADM systems often copy data between different components of the system, thereby using disks to buffer the data. Additionally, multiple components perform similar steps. One example is text extraction, necessary for content-based classification as well as for full-text indexing. Thus one focus of our approach is to reduce unnecessary copying and duplicate work in order to increase the system throughput.

The ingest service implements a crawler-style capturing of the e-mails. It actively scans mailboxes on e-mail servers for new e-mails eligible for being archived. This procedure works for regular user mailboxes, but can easily be extended to support journaling mailboxes. A direct interception of e-mails in form of an SMTP-relay is only rudimentarily implemented. It receives the incoming e-mails and stores them in temporary files before they are sent to the archive in batches.

The internal architecture of the ingest service is shown in figure 4.2. The top part of the figure shows how the e-mails are retrieved from the e-mail server. With every new job the ingest service receives a reference to a mailbox together with the necessary credentials. Typically dedicated archiving credentials are used to connect to the e-mail servers. Additionally, the ingest service receives the selection criteria that have to be applied to the mailboxes. For example "select all e-mails that are older than three months" or "select all e-mails larger than one megabyte".

To hide the details of protocol and format of different e-mail servers, we use the Java Mail API [Sun00] from Sun. This API offers methods to connect to e-mail servers, to retrieve e-mails from them, and to send e-mails. The e-mails are represented as possibly nested parts with headers consisting of simple name-value pairs. Thus they are quite close to standard Internet e-mails as specified in RFC2822 [Res01] together

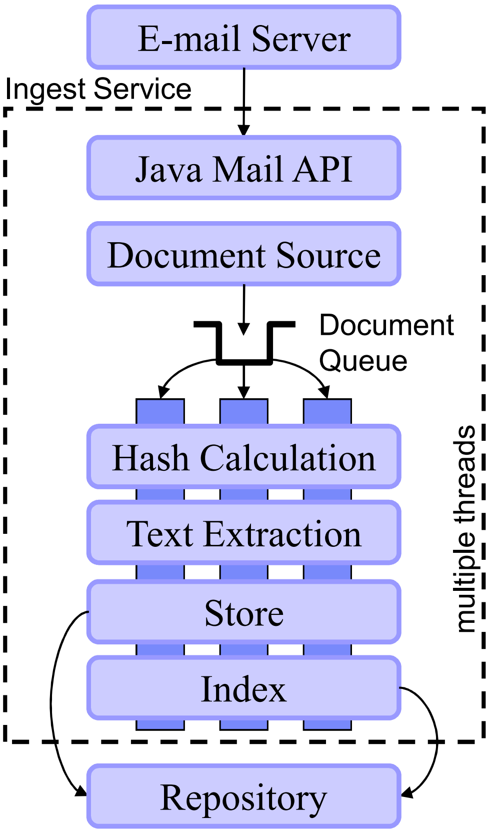


Figure 4.2: Architecture of the ingest service

with the MIME-extensions in RFC2045 [FB96b], but they could also represent other e-mails.

So-called providers implement the server-specific parts. Providers for the common protocols POP, IMAP and SMTP are already shipped as part of the API, and many others for different e-mail servers are available. Unfortunately, no provider supports Lotus Domino. As this e-mail server must be supported, a purpose built provider was implemented. It implements the Java Mail API upon the legacy Java API provided by Lotus Domino. Only functionalities used by the prototype are implemented.

The document source is another abstraction that allows to retrieve documents from other sources than e-mail servers. Another rudimentary implemented source scans a file system for documents. But we stick to e-mails for the further discussions.

After a new document is retrieved from the source it is queued up in an internal queue of the ingest service. At this point the documents are only lightweight stubs that are quickly returned from the e-mail server. As the following steps in the ingestion process can be quite computationally intensive, they are parallelized and processed by a customizable number of threads. The synchronous phase to retrieve the document is very short in contrast to the time-consuming processing phase. Thus the parallelization of the processing phase allows scaling-up the ingestion process on multiprocessor machines quite well.

Each thread picks the next available document from the single queue shared between all the threads. For each document the thread then calls a customizable set of handlers. In figure 4.2 there are only 4 handlers, but there can be several more:

Hash Calculation: At first content-based hashes are calculated for the whole document and for the different parts. This hash is used to identify the document within the system and to check whether the document or parts thereof already exist. The current implementation uses an MD5 algorithm. For a production system this is too weak as it is nowadays possible with enough computing resources to generate documents with any desired identifier. But this is unacceptable for an archive as it would open ways to tamper the archive. A better algorithm with a longer key would be desirable, but one of the back-ends currently used does not support larger identifiers. Thus for the prototypical system MD5 is used.

Deduplication: The identifier of the document and its parts (the body and attachments in the case of an e-mail) are looked up in the repository. First we check whether the complete document is a duplicate, and if this is not the case, then we check whether individual parts are duplicates. If one of the identifiers already exists the associated part of the document is flagged as a duplicate. Subsequent handlers can use this information when processing the document. For example storing the e-mail can be skipped when it already exists in the archive.

Text Extraction: Several processing steps may need a plain text version of the document. This handler extracts the plain text from the document and makes this information available for subsequent handlers. In some cases this may only require a conversion of the character encoding. But as attachments can be of any format, a large set of converters is necessary. Apache Tika[1] is an open-source toolkit that provides this functionality. The Outside In Technology from Oracle[2] is a commercial solution that supports several hundred file formats.

Classification: In many situations it is not sufficient to simply archive all the e-mails in the same way. Based on some characteristics of the e-mail it has to be processed differently. One common case is that private e-mails should not be archived at all. Many different technologies could be used for this classification. But as the

[1] http://tika.apache.org/

[2] http://www.oracle.com/technology/products/content-management/oit/oit_all.html

text interpretation necessary for this functionality was not the focus of this work we skipped this handler so far.

Store: The documents have to be stored in the archive in a way that allows retrieving them later. It is not required to completely store each e-mail, but it must be possible to reconstruct the document from the stored parts. It is for example possible to store an identical attachment of several e-mails only once and to re-integrate the attachment when one of the e-mails is retrieved.

Index: When the document is just stored in the archive, it can only be retrieved when its identifier is known. There is no way to search for documents. The basis for search is generated by the indexing handler. It uses the extracted plain text and adds a new document to a full-text search engine. This search engine can then be used to search for documents and to determine their identifiers.

As the basis for our prototype implementation we are using the text search engine library Apache Lucene[3]. Lucene is not a complete search engine, but an open source library implemented in Java that provides an inverted index and many functions to build and use this index. More information about Lucene can be found e.g. in [HG05].

The identification and separation of the different steps within the ingestion process is the basis for a customer-specific configuration of the ingestion process. Dependent on the specific requirements the handlers implementing the steps can be added or removed from the ingestion process. Thus resources can be saved if some steps are not required.

In some situations it can be beneficial to separate a step from the ingestion service and provide it as a separate service possibly on a separate node. This was investigated in [Bis07]. Reasons therefore can be a higher performance or reduced costs. As an example assume the licensing costs per node for the software component that does the text extraction are high. When the text extraction is a part of the ingest service, the license fees have to be paid for each node with an ingest service. But these nodes also have to process the other steps and can't do text extraction all the time. In that case it is cheaper to concentrate the text extraction at one or only a few nodes, and to use them by a larger set of ingest services.

The number of threads within the ingest service allows to adjust the processing speed to the requirements of the customer. With many and large mailboxes the number of threads can be increased so that all mailboxes are archived in time. The downside of a high number of threads can be a noticeable impact on the performance of the e-mail

[3] http://lucene.apache.org/

server. In a production scenario the administrator of the e-mail server does not want the perceived performance for his users to fall below a limit. One solution is to archive the e-mails in after hours. Another solution is to monitor the current performance of the e-mail server and to adjust the number of threads so that the e-mail server still has enough capacity to serve his regular users. Thus we focus on getting a maximum archival performance, knowing that it can be reduced when necessary for some reason.

An e-mail archive has to ensure that all qualifying e-mails are archived (at least) once. This comprises storing the e-mail, converting and adding it to the index, and modifying it in some way on the e-mail server. A strict transactional model like ACID [Gra81] with one transaction for each document encompassing all steps would induce too much overhead. High ingest rates can only be accomplished with a more batch-like process. Several properties of EADM permit such an approach:

- Archiving is a background process and it can therefore be done asynchronously. It does not have to be done e.g. at the moment the e-mail arrives at the e-mail server.
- The e-mails are reliably stored on the e-mail server. In a production environment typically RAID-arrays are used for the data and regular backups are available to restart after hardware failures. The e-mail archive can rely on the e-mails being available on the e-mail server until it processes them.
- An e-mail is not modified once sent. Only at the end of the ingestion process, when all other steps were successful, the e-mails are either removed from the e-mail server or they are replaced by a small stub. Yet this is a modification the EADM system is aware of.

For these reasons an only partially successful ingestion of a set of e-mails is no problem. The e-mails are still available on the e-mail server. At a later point in time the EADM system will try to archive them again. The deduplication feature of the EADM system makes sure, that the re-ingestion of several e-mails does not result in duplicate entries inside the repository.

For regular user mailboxes a modification of the e-mails by the user is possible. But in this case the e-mail could have already been modified before the first archival attempt. Journaling mailboxes overcome this problem because no regular user has access to them. Thus the e-mails cannot be modified.

4.3 Content Service and User Interface

When the documents are stored in the archive two other important functionalities of an archive are searching for archived documents and retrieving them. The application logic is provided by the content service and the presentation by a web-based user interface. A typical workflow for searching and retrieving documents in the archive is as follows:

1. The user submits his request via the user interface.
2. The content service forwards the request to the required repositories.
3. A full-text search-engine within the repository processes the query and produces a list of hits containing descriptive information about the document and the document identifier.
4. The hits are presented to the user in a web page.
5. The user selects a document he wants to retrieve. A request containing the document identifier for the hit is sent to the content service.
6. The content service retrieves the document from the repository and reconstructs it if necessary.
7. The document is either directly presented to the user or restored into his mailbox.

For the compliance officer and other users that regularly search within large fractions of the archive, further mechanisms are required to use the search results. To this end they can be stored into case folders, where they are ready to be inspected, removed from the folder, or exported.

To evaluate our approach we ported the IBM eDiscovery Manager (eDM) to our prototype. eDM allows to search for e-mails that are stored in either IBM DB2 Content Manager or in IBM FileNet P8. This extension redirects search and retrieve requests to our prototype.

4.4 Service-oriented distributed architecture

With the previously presented single-node architecture the EADM system was setup separated into several distinct services. The main reason therefore was an improved flexibility and customizability. Exemplified by the ingest service it was shown how a scale-up on larger hardware is possible. In this section we extend this architecture

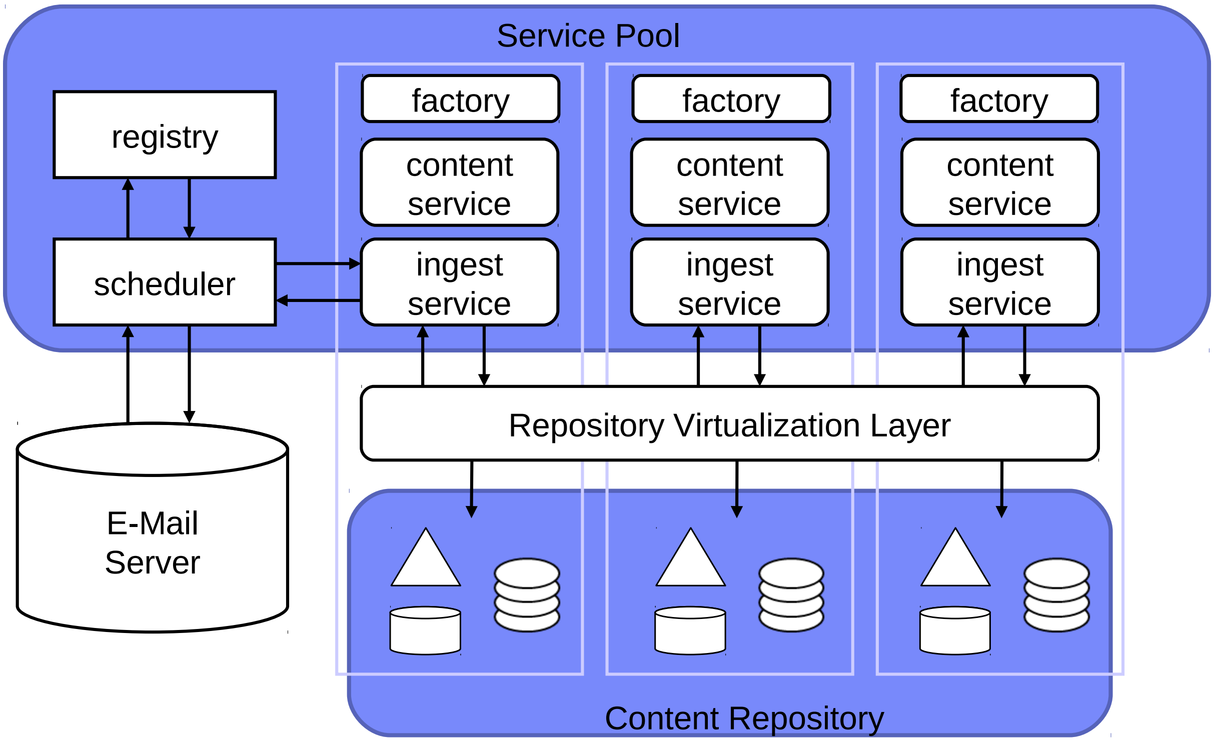


Figure 4.3: The service-oriented architecture of the EADM system

towards a distributed and service-oriented architecture that supports scaling-out on a cluster of nodes.

The services within this architecture will be Web Services that comply with the extensions Web Service Resource Framework[4] (WSRF) and Web Service Distributed Management[5] (WSDM). This allows to use standard-compliant tools to use and manage the services. One concrete implementation we are using is Apache Muse[6].

The identification of distinct services is a basis for the move towards a distributed system. The aim is to instantiate the services on different nodes as appropriate. It should also be possible to instantiate a service on multiple nodes if one machine is not powerful enough to provide the service with the required performance.

The service-oriented implementation of the EADM approach was published in [WKM$^+$08a]. Figure 4.3 shows the resulting distributed architecture of the prototype. There are basically two levels for the distribution. In the upper part of the figure the computational part of the system is distributed. On every node of the system there is one factory that allows creating instances of the other services as required. In the

[4] http://www.oasis-open.org/committees/wsrf
[5] http://www.oasis-open.org/committees/wsdm
[6] http://ws.apache.org/muse/

figure one content service and one ingest service was created on each node. But it is not necessary to instantiate each service on every machine. In the simple case all these services access one repository. This allows to scale-out the e-mail archive as long as the repository is powerful enough.

Scale-out at the application layer is also possible with existing commercial products. IBM CommonStore for example allows to set up crawlers on different nodes. Yet, all the crawlers ingest the e-mails into one IBM DB2 Content Manager system. Thus the scalability, flexibility and dynamic adaptability of this approach is limited to the characteristics of the IBM DB2 Content Manager.

Another level where distribution is possible is the data layer. In this case there is not only one content repository, but instead it is distributed over several nodes. At the repository virtualization layer these distributed nodes are integrated into one virtual repository. The aim of this approach is to scale-out the repository.

For the scaling-out of the repository it might be sufficient to add more resources at the data layer from time to time. But this very likely results in a large skew in the data distribution. The consequence is an unbalanced load distribution and thus a degraded performance. To avoid this problem another partitioning schema for the data is necessary and the data has to be arranged accordingly. This may require the movement of large amounts of data from one node to another. Partitioning strategies for the EADM system are discussed in section 5.2.

When the system already supports the re-allocation of data in order to integrate new resources, this can also be used in other situations. One such situation is when the system has more resources than necessary. Shrinking the system in such situations can lead to a more cost-efficient utilization of the data center. Another exemplary situation is if, despite of a data arrangement when a new resource joined, an imbalance develops over time. Re-arrangement of the data can then be used to balance the load again. These dynamic adaptations of the system are discussed in more detail in section 5.4.

4.5 Summary

In this chapter we first presented the single-node architecture of an EADM system. Thereby we identified the main components of such a system. In the next section we described one main component of the system, the ingest service. As it has to process all incoming e-mails it is a very CPU and disk intensive service within the system. To conclude with the main components of the system we discussed the content service and the user interface which together retrieve information from the system and present it to the user. Finally we showed how the simple architecture can be extended towards a distributed architecture that allows to scale-out the services as well as the content repository by introducing a repository virtualization layer.

5 Realization and implementation issues

In this chapter we discuss several details of the implementation of the architecture of an EADM system presented in chapter 4. First we show how the data model is implemented in a catalog using database technology and search engine technology. In section 5.2 we extend this catalog towards a distributed implementation. Section 5.3 discusses different ways to store the content in its original form and to retrieve it later on. Aspects of the dynamic adaption of the system during runtime for scale-out and load balancing on the system are discussed in section 5.4. In the final two sections we describe the setup of our test system and show the results of performance measurements.

5.1 Conceptual data model and its implementation

In section 2.2 was shown that search for the documents stored within the system is an important use case. In our example scenario the user might for example want to search for e-mails from a given user, with a given word in the subject, sent within a specific date range, containing several words in the body, or combinations thereof. To this end the system has to capture and manage the appropriate metadata about the e-mails in a component that we call the catalog.

In this section we first look at the general functionalities of a catalog. In section 5.1.2 we consider the information it has to manage in the concrete case of EADM. The following section formalizes this into a conceptual model. Sections 5.1.5 and 5.1.6 discuss the implementation of this model using a relational DBMS or a search engine. In Section 5.1.7 we present our approach with a combination of a DBMS and a search engine used to realize the catalog. Section 5.1.8 discusses the normalization of the information in the index. Finally, section 5.1.9 describes the setup of the test environment and section 5.1.10 shows the results of performance measurements.

5.1.1 EADM system catalog

The catalog plays a central role within an ECM system. It is – at least logically – a centralized component that manages the state of the system. It not only provides the above mentioned search functionality, but is also responsible for access control and lifecycle management of the documents. It primarily contains the following information:

- Specification of the items stored in the archive. It lists all the fields of the item, their names, data types and some other properties.
- Information about the actual ingestion process. The steps that have to be performed for each item and the rules that have to be applied.
- Metadata about each item in the archive. This is the data generated by the ingestion process which can be searched by the user in order to find an item. Enabling the search for relevant information that is not described exactly is an important aspect of the system.
- User and access control. Although the users may be managed externally for example in an LDAP server, it is still necessary to specify who is allowed to do what.
- The state of the system. The catalog keeps track of the existing indexes, their state, and some summary data about their content.

In the following we look at the data that has to be managed by the EADM system in more detail. Starting from an example and the standard we develop the data model.

5.1.2 Internet E-mail

In this section we describe the data format and also some aspects of the protocols of e-mails in the Internet. Although there are other, proprietary formats, the most common format of e-mails in the Internet is defined by the Internet Engineering Task Force as Request for Comments (RFC) 822 and its successor RFC 2822 [Res01]. The basic structure is very simple. The e-mail is a sequence of 7-bit characters (US-ASCII) that is

```
Message-ID: <12297377.1075855682449.JavaMail.evans@thyme>
Date: Fri, 25 Aug 2000 06:32:00 -0700 (PDT)
From: phillip.allen@enron.com
To: suzanne.nicholie@enron.com
Subject: Re: Meeting to discuss 2001 direct expense plan?
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
X-From: Phillip K Allen
X-To: Suzanne Nicholie

Suzanne,

 Can you give me more details or email  the plan prior
to meeting?  What do I need to provide besides
headcount?

Otherwise any afternoon next week would be fine

Phillip
```

Figure 5.1: Example of an e-mail according to RFC 2822

separated into lines with a maximum length of 998 characters. The e-mail is separated into two parts separated by an empty line: 1) a header with several header fields, and 2) the body. The header fields start with a name followed by a colon (“:”) and the body of the field. To get around with the line length limitation it is possible to fold a header field into multiple lines. An example of an e-mail is shown in figure 5.1.

The header fields of an e-mail are primarily informal. Most of the fields were created by the sender of the e-mail or automatically by his e-mail client. Table 5.1 lists the header fields as defined in the standard. The most popular ones are “from” specifying the sender of the e-mail, “subject”, and the different destination fields “to”, “cc” and “bcc”. It is also possible to add application-specific headers which should start with the prefix “X-”. The information in the header helps the user classifying the e-mail to some extend. The trace fields and the resent fields are prepended to the e-mail by the e-mail servers as they process the e-mail.

Field	**Min**	**Max**	**Notes**
Trace Fields			Prepended by mail transfer agents during delivery.
return-path	0	∞	
received	0	∞	
Resent Fields			Used when e-mails are reintroduced into the transport system.
resent-date	0	∞	
resent-from	0	∞	
resent-sender	0	∞	
resent-to	0	∞	
resent-cc	0	∞	
resent-bcc	0	∞	
resent-msg-id	0	∞	
orig-date	1	1	Timestamp when the sender sent the e-mail.
Originator Fields			
from	1	1	Author(s) of the e-mail.
sender	0	1	Used when e.g. a secretary sent the e-mail on behalf of the author in the from field.
reply-to	0	1	
Destination Fields			
to	0	1	
cc	0	1	
bcc	0	1	
Identification Fields			
message-id	0	1	Should be present and sender guarantees uniqueness.
in-reply-to	0	1	
references	0	1	
Informational Fields			
subject	0	1	
comments	0	∞	
keywords	0	∞	
optional-field	0	∞	

Table 5.1: E-mail header attributes

From an archive perspective the most interesting header fields are those fields that can be used to find relevant messages. Fields with information the user has in his mind when searching for e-mails and that allow restricting the set of qualifying e-mails. These are 1) the informational fields, where especially the subject is widely used, 2) the originator and destination fields that list the set of users that were able to see the message, and 3) the date field indicating the creation time of the message.

When interpreting the header fields one has to keep in mind that all the fields except for the trace fields and the resent fields are generated either by the sender of the e-mail or by his e-mail client. Thus their creation is completely in the hand of the e-mail sender and they are therefore not very reliable and trustworthy. Some examples why the header fields should not be trusted too much are the following:

- According to the standard the message-ID must be a globally unique identifier for the message. It is often composed of the e-mail address of the sender or the address of his e-mail server and a timestamp or random value. Constructed in this way the message-IDs can be unique with a sufficiently high probability. But e-mail clients are not always well-behaved. There are cases where a spam-e-mail always contained the same message-ID. If a system relies on the uniqueness of this message-ID e.g. to eliminate duplicate e-mails, false duplicates could be ignored although their content is completely different.

- A timestamp of an e-mail is important for an archive to decide when the e-mail can be deleted. But the orig-date might be off either accidentally, when the clock is not set correctly or the on-board battery of the computer is empty, or intentionally, in order to place the e-mail at a more prominent position at one end of the inbox of the recipient. If a user searches for e.g. all mails from within the last two weeks, and the archive uses the date from the header as basis, the user might not find the e-mail he is looking for. Even more problematic would be the deletion of expired messages based on this date.

- The value of the subject field clearly depends on the sender of the e-mail. But even when the initial sender of an e-mail has chosen a good subject, it is not uncommon, that the topic of the e-mail changes completely after it was replied to or forwarded several times.

- The e-mail-addresses in the originator and destination fields also have their challenges. First of all they are only informational. The information that is used to deliver the e-mail is part of the protocol that transfers the e-mail, e.g. the Simple Mail Transfer Protocol (SMTP). This information does not have to match the information in the header at all. Spam e-mails often exploit this fact. Additional challenges arise from the ambiguity of the information. A user may have several mailboxes, and there can be several aliases for one mailbox. So when looking for e-mails from a specific user one has to search for all the different names. Additionally, with e-mail distribution lists the recipients are not mentioned explicitly in the destination fields. Resolving these lists to the individual users is realistically only possible for the e-mail server at the time the e-mail is delivered. But this information is not recorded in the e-mail and therefore not available.

In summary an e-mail archive should provide search capabilities on the header fields as they may contain valuable information. But the system should not solely rely on them. It is therefore necessary to support search capabilities on the body and the attachments, too. Furthermore the archive should not exploit any of this information for internal processes without validating it against other sources of information.

So far we discussed only a very simple form of an e-mail with one body in US-ASCII. Nowadays this is no longer sufficient for most e-mails. Text with different character sets or formatting, and attachments also have to be transferred. In these cases RFC 2822 is merely a container, like the envelope of a normal mail. How to map the new requirements into the old e-mail is specified by the Multipurpose Internet Mail Extensions (MIME) [FB96b, FB96c, Moo96, FKP96, FB96a].

The most important extension is the introduction of parts. They are encoded in a way such that the e-mails with multiple parts are still compliant to RFC 2822. Thus for software components like the e-mail server, which do not care about the content of the e-mail, this is transparent. But when the (optional) “Content-Type” header starts with “multipart/”, the lines of text in the body can be interpreted by the e-mail client as multiple parts. A boundary separates different parts. Each part starts with its own header followed by the body of the part. How the body of the part should be interpreted is defined in its header. It could be e.g. a simple text, an image, or yet another multipart.

There are three predominant variants of multiparts: alternative parts, mixed parts and related parts. Alternative multiparts are e.g. used to wrap different renditions of the body. Many e-mail clients nowadays produce HTML as the body of the e-mail. But not every e-mail client is able to or allowed to process the HTML. Therefore an alternative part with a plain text version of the body is often generated, too. Both parts are packaged in a multipart with content-type “multipart/alternative”. The e-mail client of the recipient can choose which part to present to the user.

Multiparts with content-type “multipart/mixed” are mostly used for attachments to the e-mail. The content-type “multipart/related” is used when the parts belong together and should be used together. A common example are e-mails with an HTML body where images included in the page are also shipped as parts of the e-mail.

Other aspects of MIME are e.g. the transfer encoding and the character encoding. The transfer encoding defines how binary content is transferred within an e-mail that only allows using 7-bit characters. Often the Base64-encoding defined in the MIME-standard is used, where only 64 different characters are used and thus the size increases by 33%. The quoted-printable encoding is an alternative when the part is nearly ASCII-text and only a few characters have to be quoted. The character encoding defines how the different characters of a language are encoded (e.g. UTF-8).

With the introduction of MIME, e-mail got a lot more usable for many persons, but it also got more complicated for e-mail clients or an archive to interpret them. To retrieve the plain-text content that is required for full-text search, several pre-processing steps have to be performed

- Separating the e-mail into the different parts
- Deciding which part to use as the body
- Decoding the transfer encoding for non-7-bit-data (e.g. Base64)
- Mapping the character encoding to a common encoding (e.g. UTF-8)
- Extracting compressed archives like ZIP-files, . . .
- Converting attachments in application-specific formats (e.g. Word, PDF) to plain text
- Fetch external information referenced by the e-mail. Sometimes e-mails don't have all their images included, but instead only have references to copies stored on some web server. Or the e-mail references documents in a team room or on a web server. They can be seen as part of the e-mail, although they were not transferred with it.

5.1.3 Conceptual data model for EADM

In this section we discuss different alternatives to create a data schema for e-mails. The previous section already indicated that there are at least two alternatives: 1) The e-mail is just one entity with several header fields and one body as defined in RFC 2822, or 2) the e-mail has a body that may consist of multiple parts as defined by the MIME standard.

The different models are all valid representations. The difference lies in the interpretation of what is considered the primitive, non-decomposable building block.

When developing a schema for a database application one aim is to achieve some normal form in order to reduce redundancy and to increase consistency. The first

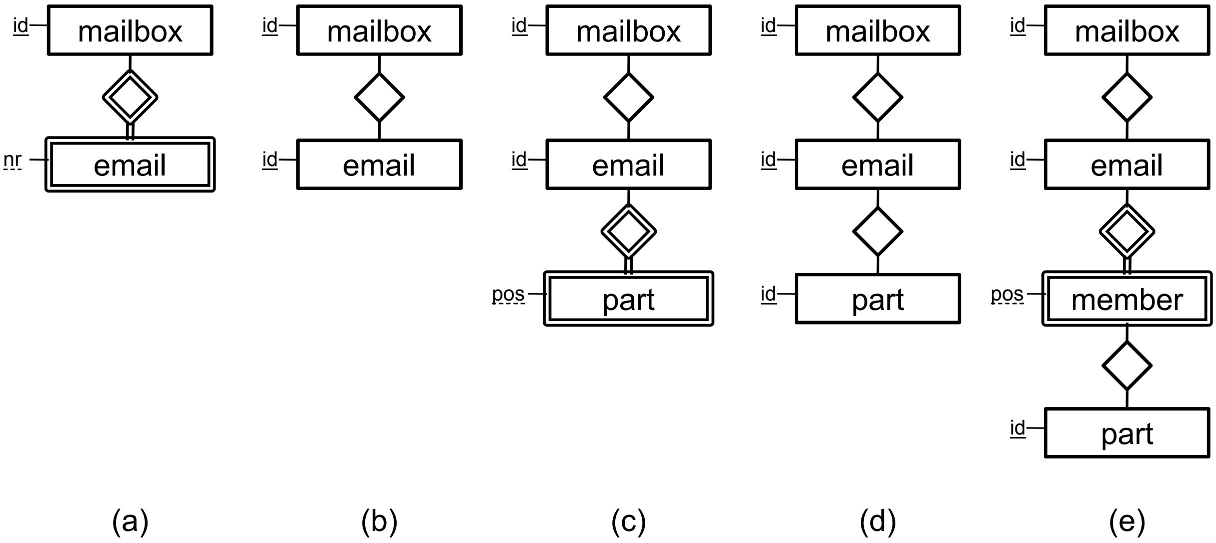


Figure 5.2: ER-Models for E-mails

normal form (1NF), which is the basis for all other normal forms, enforces that the elements in a domain are atomic/nondecomposable [Cod70]. Whether some value is atomic can be ambiguous. A date for example is typically assumed to be atomic. But it is possible to split it up into a year, a month and a day. If the application e.g. often only uses the year in queries it might be more appropriate to look at the date as a non-atomic value, and to store it as three separate values. Similarly, a string value could be seen as non-atomic when e.g. a prefix or another arbitrary sub-string has a special meaning for the application, e.g. when the first characters of a department identifier denote the location of the department.

After deciding which values are atomic, the other normal forms decrease the redundancy and increase the consistency of the data model. Depending on the choice of the atomic values, the resulting data model consists of more or less entity types. Several approaches to model the e-mail are shown as entity relationship diagrams in figure 5.2. At this point we focus on the rough structure of the e-mail and ignore most of the properties like e.g. the subject of the e-mail.

In figure 5.2(a) the mailboxes are modeled as strong entities and the e-mails are weak entities that belong to a mailbox. They are identified by e.g. the position within the mailbox. In other words the mailbox contains the list or set of its e-mails. This model results in a high storage consumption as every mail in each mailbox is a different entity.

In figure 5.2(b) an artificial unique key is assigned to the e-mails so that they now are strong entities of their own. This allows to reference an e-mail from multiple mailboxes and thus to reuse already stored e-mails. Now the tricky part is the generation of the unique key. When archiving a new e-mail it must be possible to check whether the same e-mail has already been archived. Comparing the new e-mail with all archived e-mails byte-by-byte is clearly infeasible. The message-id within the e-mail header would be ideal, but as discussed in 5.1.2 it is not reliable enough. A good way to generate the key is using algorithms that compute content-based identifiers like MD5 or SHA-256. These algorithms generate a short (128 or 256 bit) summary of the content, where the same content always produces the same identifier and there is a very low possibility for collisions.

In the previous two models the e-mail was modeled as in RFC 2822 where there is only one body with no internal structure. In figure 5.2(c) we also represent the parts of an e-mail within the data model. In this case they are represented as weak entities assigned to their enclosing e-mail. As the recursive structure of the parts is not important for the archive they are flattened to a simple list. But it could also be represented either with a separate relation or by employing a hierarchical numbering scheme.

In praxis it is not uncommon that one attachment occurs in several e-mails. For example when spreadsheets, text documents or slides are sent from one person to another and then forwarded to another group of persons and so on. To support the re-use of the parts they are also modeled as strong entities in figure 5.2(d). Again a content-based identifier, this time calculated only over the individual parts, is used as the primary key. The position of the part within an e-mail is stored as a property of the relation within e-mail and part.

In contrast to the previous model figure 5.2(e) separates a part into a body and a header. The motivation for this separation is that a file that was attached to several e-mails may have different headers in these e-mails. E.g. when it is referenced with a different filename. Of course there is a trade off whether this separation pays off or not. The header labeled "member" is a weak entity that belongs to the e-mail, and the body is a strong entity.

In this section we discussed several ways to represent the e-mails in a data model. From a simple model, where the complete e-mail is just one entity, to a more complex model, where e-mails are shared between mailboxes and attachments are shared between e-mails. Yet there is no clear winner. It depends on what the application wants to do with the e-mails. Therefore different approaches are followed in later sections when the data model is implemented.

Besides information that is contained within the e-mail and can be extracted from it, the archive also has to deal with information necessary for the management of the archive in order to perform the use cases discussed in 2.2. A simple model with the most important entity types is shown in figure 5.3. The modeling of e-mail and part is not important in this figure and is just an arbitrary choice.

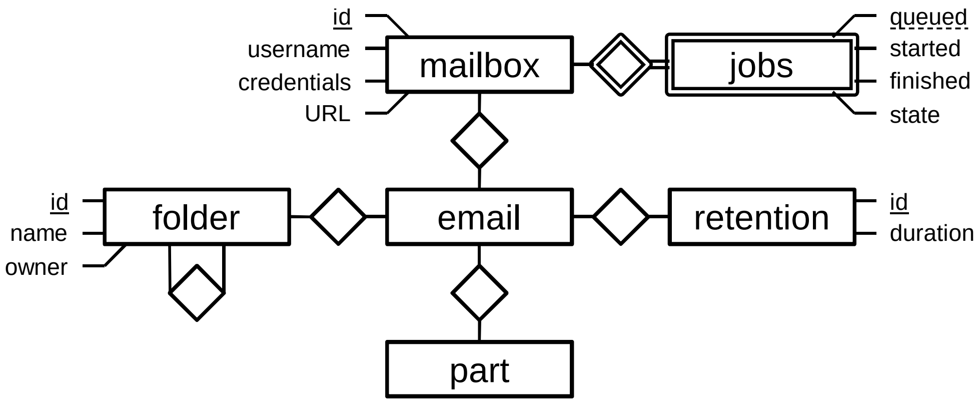


Figure 5.3: ER-Model showing entities with relationships to the e-mail

Especially in the case of mailbox management the e-mail archive does not only archive the e-mails it gets from somewhere, but instead it actively retrieves qualifying e-mails from e-mail servers. The necessary information to do this is recorded in the mailbox and in job entities. The mailbox entity represents one mailbox on an e-mail server and contains information like the name of the user owning the mailbox and a URL as well as credentials required to connect to the mailbox. A scheduler regularly iterates over the list of mailboxes. Based on specific rules the scheduler decides which mailboxes have to be archived and adds a corresponding entry in the jobs table. This table is the input queue for crawlers that actually process the mailboxes.

When the e-mail archive has to process all e-mails there are two options to capture the e-mails. Option one are journal mailboxes maintained by the e-mail server. In such an environment a new entry is added to the jobs table every time the current journal mailbox is full and the e-mail server starts with a new one. Thus the further processing is similar to the previous case. Option two is introducing an SMTP-gateway that receives all e-mails and forwards them to other systems as needed. To this end we implemented a simple gateway that stores the arriving e-mails in a temporary file. Similar to the previous case an appropriate entry is placed in the jobs table when the temporary files reach a customizable maximum size. The further processing remains the same. Thus the pre-processing differs for the three cases, but they all end up as entries in the jobs table.

After the e-mail is archived, the next question is how long does it have to be archived? In praxis a wide range for this retention period can be seen as discussed in chapter 2. The retention period may even vary within one company. E.g. it might be necessary to keep the e-mails from upper management or the financial department for a longer period than for other employees. Therefore several retention categories can be necessary, each having its own retention period. During the ingestion process the appropriate retention category is assigned to each e-mail. In the background the system regularly scans the archive for e-mails that can be removed.

In the compliance scenario as discussed in 2.2 the compliance officer, lawyer, etc. has to work with large sets of e-mails. Therefore they need mechanisms to save result sets of a search and to modify or export them. This is realized in figure 5.3 using the folders. The hierarchical structure of the folders allows to organize them as appropriate and e-mails can be assigned to multiple folders at a time. A user declares an e-mail as relevant to a specific case by adding it to the case's folder. As long as the e-mail is relevant to a case it must not be deleted from the archive. Therefore every e-mail that is assigned to a folder is implicitly put on a "legal hold". In this state an e-mail is not deleted, even when its retention period is over.

Looking at the data volumes the metadata extracted from the e-mails is by orders of magnitude the largest part of the total information. Also the rate of inserts or updates is much higher than for the others. For each e-mail that is archived a new entry is added. And this typically happens more often than everything else in the system. Thus the efficient management of this data is crucial for the EADM system.

5.1.4 Catalog implementation alternatives

After defining the conceptual data model of the application, the next step discussed in this section is the identification and selection of technologies to implement it. We see two alternative technologies for the implementation of the catalog: relational database management systems (DBMS) and search engines. In the following we briefly describe them and discuss their advantages and disadvantages for an EADM system.

5.1.5 Relational Database Management Systems

Relational DBMS are based on the relational model introduced by Codd in 1970 [Cod70]. In this model information is represented as sets ("relations") of n-tuples and queries are expressed in a declarative language based on the relational algebra. The big benefit of using an abstract mathematical model is the separation of the application from the actual implementation of the data model ("data independence"). It allows optimizing the storage of the data on the disks without having to change the application. Thus the application and the DBMS can be improved and extended independently.

In a relational DBMS the data is typically stored in a normalized form. The aim of this normalization is to decrease redundancy and to increase consistency. The result is a set of tables that is interconnected via primary keys and foreign keys.

The most common language to query relational databases nowadays is SQL [SQL03]. It is a declarative language based on the relational algebra. At the core it is a readable language to query relations.

Although relational databases were originally targeted towards structured data, they can also be used to handle unstructured, textual data. Yet, a string (or varchar in SQL) is often not just an atomic value. It contains distinct words or even has an application-specific structure. Standard SQL supports text search within strings by means of the LIKE predicate, which, however, is limited to sub-string-search with wildcards. The following example retrieves all rows from a table called “email” where the character sequence “meeting” occurs somewhere in the title.

```
SELECT * FROM email
WHERE subject LIKE '%meeting%'
```

To support search in large text fragments better, and to provide a richer query language for textual data, many DBMS offer text search extensions. Examples are the IBM DB2 Net Search Extender[1], Oracle Text[2] and OpenFTS[3] for PostgreSQL. With these extensions, additional full-text indexes can be created for selected parts of a table. One problem for such systems are queries that contain structural and full-text predicates. Imagine a query requesting all mails from 1996 where the body contains the word 'bargain'. The first predicate has to be evaluated on the tables, and the second one using the full-text index. On a large archive the query for mails from 1996 will create a large result and the search for the word in the body as well. Due to the large intermediary results of the two systems, the calculation of the intersection is expensive, too.

With a normalized data model, a user transaction often touches multiple types. The transaction reads and writes the data at different points in time. With multiple concurrent users and no coordination mechanism this will result in an inconsistent database sooner or later. Also in the case of an error some parts may already be persisted to disks and others not. To avoid these problems DBMS provide a transaction mechanism that groups operations belonging together into one transaction. For these transactions it guarantees the so-called ACID-properties [GR93]:

Atomicity: A transaction is either executed completely or is not executed at all. It is not possible to sub-divide the transaction into smaller parts and to execute only some of the parts successfully.

[1] http://www-306.ibm.com/software/data/db2/extenders/netsearch/

[2] http://www.oracle.com/technology/products/text/index.html

[3] http://openfts.sourceforge.net

Consistency: The transaction transforms the database from one consistent state into a new consistent state. The DBMS e.g. has to make sure that unique keys, foreign keys and other constraints are still valid.

Isolation: Each transaction logically has exclusive access to the database.

Durability: When a transaction is successfully completed the DBMS has to persist the changes and has to make sure that they are not lost.

In a local system these properties are often ensured by utilizing locks and logs. This is an overhead that needs some resources. In a distributed environment the different nodes participating in a transaction have to be coordinated. This means yet another overhead. A protocol commonly used therefore is the two-phase commit protocol.

In an environment with many concurrent users, the ACID-properties are quite strict. The isolation property does not prevent parallelism completely, but ensuring the logical impression of exclusive access requires some synchronization overhead. The overhead can be adjusted to some extend by specifying the necessary isolation level for read operations.

The CAP principle [FB99] mentions a boundary for a distributed system. From the three properties strong consistency, high availability and partition-resilience one can pick at most two. All three properties are nice to have, but when implementing a system one has to find a compromise. For a DBMS the consistency is very important, and thus the availability is weaker. Other systems accept a weaker consistency in favor of a higher availability.

Another drawback of the strong consistency model guaranteed by the ACID properties is that the performance penalty that has to be paid to ensure the consistency is very high. Thus the question is whether the system has to be that strict? An easement that is often tolerated for an EADM system is that newly added documents do not have to be retrievable immediately. As long as the system converges to a consistent state without losing any data, a delay of up to a few hours may be tolerated. This can be used to reduce the synchronization overhead when ingesting new documents and thus to increase the performance.

Several other features of a DBMS are valuable for EADM, too. Backup and recovery for example are necessary for any production system and with adequate management of buffers, I/O and queries, the systems ensure efficient and resilient operation even under heavy load generated by many concurrent users.

Relational databases are used by existing e-mail archiving products to implement the catalog. IBM DB2 CommonStore for example stores its data within the IBM DB2 Content Manager, which in turn uses an IBM DB2 Universal Database or an Oracle database as its back-end.

5.1.6 Full text search engines

The major objective of search engines is to enable search on unstructured documents or documents with a simple structure. Most prominent examples are Internet search engines like Google[4], Yahoo![5] and Bing[6]. The user has two main requirements for a search engine: 1) it has to determine the results quickly, and 2) the result must have a good quality. Both requirements make great demands on the search engine. On the one hand the large collections of data that are often covered by a search engine require heavily optimized data structures and algorithms to reach the performance goals. On the other hand the user often does not know which documents exist in the system and especially which documents he wants. Instead he has information needs that he somehow has to express in the query language of the search engine. The search engine then produces a list of the documents that according to some measure are most relevant to the query. Thus the quality of the result depends on how precisely the average user expresses his information needs in the query language, and on the matching and ranking of the relevant documents by the search engine.

Compared to a relational database management system a search engine has a very simple data model: a set of documents. The documents might be e.g. files in the file system, web-pages or e-mails. For a search engine a document is merely a sequence of terms extracted from the original document. Some search engines support a simple structuring of the documents where each document has a set of named fields. In that case the value of the fields is a sequence of terms. By adding the appropriate fields during ingest it is for example possible to explicitly search for documents from a specific author or with specific title, and not just for documents that contain the name of the author or the title at some position.

As discussed in the previous section the data model for a DBMS is often normalized to decrease redundancy and to increase consistency. This is not the case for search engines. In that case the data model is highly denormalized. In fact there is only one type, the document, which contains all the information relevant to this document. One reason is that during query processing it is not necessary to combine the results from different types. In practice there can be different types of documents in the search engine, each with a specific set of fields. But for the search engine this is transparent and they are all just documents with some fields. Furthermore, most often the expected result from a search engine is a set of documents, and not some partial information like the set of names of authors of an article with a given word in the title.

A search engine also does not support references between objects. There are no pri-

[4] http://www.google.com

[5] http://www.yahoo.com

[6] http://www.bing.com

expression	**matches documents with...**
compliance	the term compliance in the default field
subject:compliance	the term compliance in the field subject
+bargain +opportunity	the two terms bargain and opportunity in the default field
+tomcat -apache	the term tomcat but not the term apache in the default field
(money OR rich) AND friend	the term money or the term rich, and the term friend in the default field
“sure thing”	the phrase “sure thing” in the default field
legal*	a term in the default field that starts with “legal”
maier˜	a term that is similar to the name “maier” in the default field
date:[20010101 TO 20011231]	documents where the field date is within the year 2001

Table 5.2: Sample expressions for the Apache Lucene query parser

mary key fields or foreign key fields like in relational DBMS. Yet the application using the search engine may interpret some fields as keys. E.g. Google uses hyperlinks between the documents when calculating its page rank [BP98]. But the search engine does not enforce such references. By way of the denormalization of the data model most references are contained within one document. The lack of ensured consistencies can be a performance benefit as the engine does not have to ensure them.

With the simple structure of the data the query language of search engines is also comparatively simple. It only has to support the selection of qualifying documents. Typical constructs are the terms and in which field they have to or should occur, boolean operators to combine queries, proximity searches to find terms nearby, and fuzzy or wildcard queries. But there are for example no joins between different entity types. Some of the expressions recognized by the query parser from Apache Lucene are shown in table 5.2. If no explicit field is specified, a default field will be used, e.g. the field with the text from the body of the e-mail.

One of the main challenges in the area of information retrieval (IR) is that there is often no 100% correct answer to a query. Often the user does not look up all the documents with a given keyword, but instead of that he has more abstract information needs that he wants to satisfy. Due to different word forms and synonyms a document may be the one the user is looking for, although the exact term as specified in the user's query is not part of this document. Thus, important features of search engines are language-specific normalization of the terms and further enrichment of the set of terms during the ingest. Frameworks like UIMA [7] and LingPipe [8] provide a foundation to incrementally derive more structured information that augments the information from the documents. Of course the processing of the queries has to match these modifications. E.g. when words are reduced to their stem when building the index this reduction should also be applied to the queries.

Finding all relevant documents is only the first step for the search engine when processing a query. An equally important step is the ranking of the documents so that the most relevant ones are shown on the first page. Whereas databases only allow to sort by one of the structured fields, search engines try to rank the results according to the relevancy for the query. For example the TF-IDF measure (see e.g. [BYR99]) calculates for each term in the query how relevant it is for each document and weights it in relevance of the term within the complete collection. Of course professional search engines like Google and Yahoo incorporate many more factors into their ranking function. Although these ranking algorithms are not optimal, the result is often much more usable than having no or some arbitrary order.

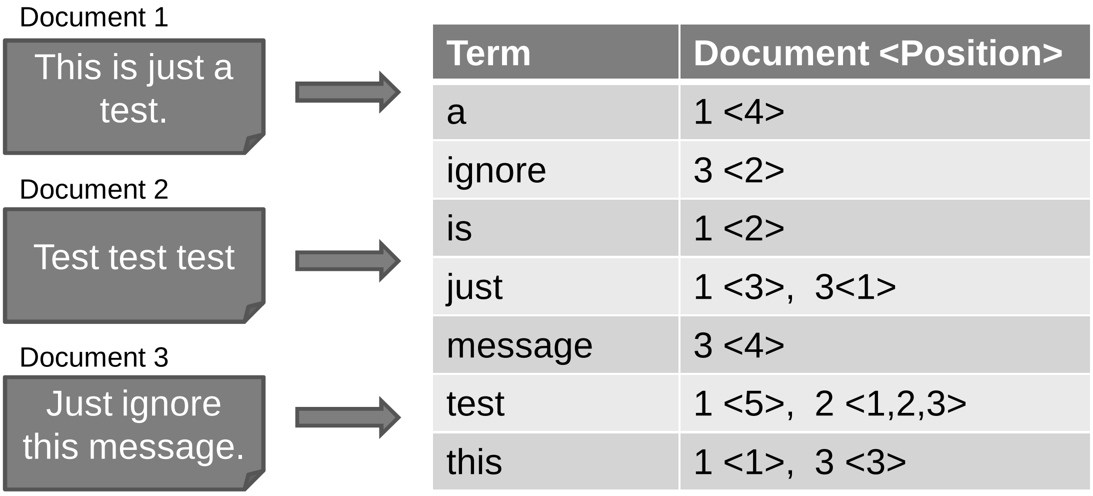
Internally search engines often rely on an inverted index. When documents are added to this index, first a sequence of tokens is extracted from the documents. After normalization steps like removing stop-words, case-unification and stemming, the lists are "inverted". That means the terms are sorted alphabetically and for each term the identifiers of the documents that contain this token are recorded together with the positions of the token within the documents. An example is shown in figure 5.4. The inverted list is stored in a compressed format that is optimized for fast query processing. To find the documents that contain a given word, this word can be efficiently searched within the inverted index. The document identifiers are attached to this word. When there are multiple words in the query, the list of document identifiers for each word has to be retrieved and then, depending on the query, the union or intersection has to be computed to produce the result. The position information can be used for proximity queries or phrase queries.

The downside of optimizing the inverted index focused to the search functionality are updates of the collection as a whole and of individual documents. Lester e.a. [LZW04] examines three main strategies to maintain an inverted index:

1. The index is amended in place. Thus for every word in a new document the new document has to be appended to the list of documents containing this word.

[7] http://www.research.ibm.com/UIMA/

[8] http://alias-i.com/lingpipe



Term	Document <Position>
a	1 <4>
ignore	3 <2>
is	1 <2>
just	1 <3>, 3<1>
message	3 <4>
test	1 <5>, 2 <1,2,3>
this	1 <1>, 3 <3>

Figure 5.4: Inverted index

Since a document contains many words a lot of disk operations are required. This can be alleviated to some extend by adding multiple documents at once.

2. The index is rebuilt from scratch. Of course this is hardly more efficient for individual documents than the previous strategy. But this strategy is in fact used when e.g. the index only has to be updated in long regular intervals.

3. The third strategy is to internally use multiple inverted lists. New documents are first added to a new one. These inverted lists are then later on merged into one inverted list.

All three strategies have in common that they are not efficient for many small transactions. Thus an online transaction processing (OLTP) workload is not supported efficiently by an inverted index. In practice the inverted index is thus often updated by an asynchronous background process. These batch updates of multiple documents can then be performed more efficiently.

On the one hand, the separation of ingest and search allows to optimize both operations, ingest and search, independently. This at least has the potential to result in a faster system. On the other hand, this model loosens consistency. New documents are not immediately searchable with the search engine. When a just archived document cannot be found, the confidence of the user into the archive might decrease. Yet for an archive a lag of a few hours is often tolerable. Especially when the e-mails are archived by a crawler in the background and are not actively archived by the users. The maximum allowed lag is a parameter of the system that allows to weight ingest performance and consistency.

Deletion of documents is another operation that is not supported efficiently by an inverted index. In a first step one has to iterate over all the tokens in the index and check whether the document is in the list for the token. In that case the document has to be removed from this list. This is even worse than adding a new document, because in addition to index updates the complete index has to be scanned once to find all the references to the document.

To alleviate the problem with delete operations, search engines like Apache Lucene record a separate list with documents that are deleted. In this case the deletion itself is merely flipping one bit and persisting this change. When searching for documents only the documents that are not marked as deleted are presented to the user. Thus the deleted documents still have an impact on the performance of the search engine. As a consequence, the inverted index should be cleaned up from time to time.

A general purpose ECM system definitely requires a delete operation. For an EADM system this is different. Deletion of e-mails is not done interactively on a user's request. Instead the documents are deleted when they exceed their retention period, and when they are not on a legal hold. Thus, delete is not a response-time critical operation, and can be performed by the system as efficiently as possible in the background. Nevertheless, after several years every e-mail once ingested will be deleted.

Updating the document collection by deleting old version of documents and inserting new ones is supported by search engines. But what about updating individual documents? The inverted index does not efficiently support determining what belongs to one document and so the same problem as with deleting documents arises. Again an inefficient scan of the complete index has to be performed in order to determine all the terms that refer to the old document. A further handicap is the fact that the implementations of the inverted lists are often stored in a compressed form that does not have free space for changes. One possible approach to support updates is to mark the old document in the index as deleted and to ingest a new version. A precondition for this approach is that all the information is still available. E.g. it is not possible to update only the subject-field if the information stored in the other fields is not available. Either the complete information is provided with the new document, the missing information is stored somewhere, or it can be re-extracted from the archived document.

At a first sight the deficiency with respect to update operations is not a problem for EADM. E-mails once sent are never updated. But some aspects associated with the e-mail may change: for example the assignment to folders, or the authorization information. This has to be considered appropriately.

A general update of indexed documents in an inverted index is not supported efficiently. Yet several applications could benefit from the possibility to update at least a small part of the document. Examples are adding and removing semantic tags to a document, including current status information of the document and implementing role-based access control.

One approach to support updateable fields better is to add this functionality on top of the inverted index. It is for example possible to store these fields in a DBMS and to join the results from the DBMS and the inverted index when processing queries. The problem with this approach is that the intermediary results from the two systems can be quite large, and thus the processing of the join can be expensive. Especially in the general case when both intermediary results have an impact on the set of final results and their ranking. In the special case when only one system determines the result set and its ranking, and the other system fetches only additional information to be displayed to the user, the execution costs are manageable. An example is when the aforementioned tags are displayed to the user only next to each hit. But when these tags are also part of the query, the general case has to be applied.

To support small updateable parts of the documents Oracle Text in release 10g introduced so-called MDATA-sections. The metadata contained in these sections underlies a few size restrictions, but it is updateable and efficiently searchable within the search engine. The "tag index" implemented by Jason Rutherglen for the OceanSearch project adds a similar feature to Apache Lucene. It manages the updateable fields in a separate data structure that can yet be efficiently exploited during query processing. The key is to maintain this additional data structure in line with the index, so that no general purpose join is required to combine the results. An important aspect thereof is to keep the internal document identifiers in sync even when the index is merged, optimized, or documents are deleted.

With the simple data model and the simple query language provided by a search engine the options when implementing the conceptual model of the EADM system are limited. Basically all properties that might be relevant for some queries have to be integrated into one document. Therefore the necessary relations are traversed during the ingestion and are stored in the relevant document in a denormalized form. The missing join operation during query processing is thus bypassed by materializing the join result during the ingest. To this end the inverted index is comparable to a materialized query table. This of course leads to a larger storage footprint. The advantage is a simpler and faster search process as long as the additional storage requirements are moderate.

In general, the user of an information management system prefers every piece of information being stored only in one place. E.g. there should be one place to change the address of a customer, and the user should not have to change it for every account of the customer. Thus the information should be stored in a normalized form. But this is not the case for an EADM system. It has to persist the original information contained in the e-mail. As an example the change of the e-mail address of a user must not change the address in already archived e-mails. Thus the denormalization enforced by the simple data model of the search engine is to some extend also reasonable for an EADM application.

An example of a system that uses a search engine to implement the catalog is the Symantec Enterprise Vault [Sym06]. It uses search-technology from AltaVista for its catalog.

5.1.7 Combining DBMS and search engines for EADM

As proven by the two exemplified systems, both technologies can be used as part of an ECM system to implement the catalog. Both approaches have their advantages: databases are more robust and reliable, and with the appropriate extensions searching for text within the data is possible. Search engines provide a faster and more effective search on textual data, and their less stringent consistency can be a performance benefit.

The reason why we discuss both technologies is that we would like to have features from both of them. Due to the large fraction of unstructured textual data and imprecise queries from the users, the search engine is the favorite. But there are also parts of the data model that have to be updateable. Thus we prefer a combined approach described in this section. Many approaches to combine database systems and search engines have been proposed. A few examples are [Bre04, CDY95]. The focus is to come up with enhanced techniques for searching equipped with specific solutions for ranking the search results. In contrast we are not aiming at a general purpose integration. Our approach exploits the peculiarities of EADM when combining the two systems. The aim is not to come up with yet another integration of relational databases and search engines, but instead to come up with an approach designed specifically for EADM, or possible nearby applications.

The main pitfalls when combining two distinct information management systems are the execution of queries involving both sources and the synchronization of updates. There will be at least one set of keys that correlate entities in one system with entities in the other system. When executing a query the results of the two systems often have to be combined with an equi-join. Especially in the area of information retrieval the intermediary results for parts of a query can be very large, although the final result is small. As an example we look at a query to retrieve all e-mails where the sender is “frank” and the body of the e-mail contains “heathrow”. Now if each clause of this query is processed by a different system, then one system would produce a list of all e-mails from “frank”, and the other system would produce a list with all e-mails that

	mutable	**immutable**
searchable	• Relation mailbox - email • Relation email - part	• extracted text from headers, body and attachments • mailbox owner
not searchable	• mailbox url and credentials • folder content • retention information	• archived documents in original form

Table 5.3: Searchable vs. mutable parts of the data model

contain “heathrow” in the body. Depending on the data collection both of these lists can be very large. In a final step both lists have to be joined on a common e-mail identifier. The final result may contain only a handful of e-mails. The user then only sees a few results, and wonders why it took that long.

The above example shows, that the way how the data is mapped onto the different systems, the partitioning, can have a big influence on the performance of the system. The more independent the different systems are used, the lower is the overhead to integrate and synchronize them. This, of course, is dependent on the workload the system has to sustain.

Table 5.3 classifies the different aspects of the data model with respect to two categories: mutable and searchable. Mutable attributes of an entity are those attributes that may change over the lifetime of the entity. Searchable attributes are those attributes that may occur in search queries from the user. Of course it is possible to also retrieve the not searchable attributes, but they will not be used as part of a query when searching for documents.

In the upper right corner of the table is the immutable and searchable information. Regarding the data volumes the largest fraction of the data is allocated there. All the information extracted from the e-mail, the headers like subject, sender, and recipients as well as the plain text extracted from body and attachments belongs into this corner. The name of the mailbox owner is also in this corner. It is used in queries to either restrict the query to a few mailboxes, or to enforce access control restrictions.

The information in the upper left corner correlates the mailboxes to the e-mails and the e-mails to their parts. Both relations may vary with time: 1) A new e-mail can be added to a mailbox, 2) an already known e-mail can appear in another mailbox too, 3) an already existing part can occur in a new e-mail. These relations can be queried directly, yet they have to be taken into account when processing a query with restricting clauses on different types.

The lower left corner contains mutable information that is not searchable in the sense defined above. This information is queried in separate use cases and shown in different views. A folder for example contains persisted hits from a previously executed query. A compliance worker looks at a folder and refines its content when working on a case. After persisting the query he only works on the folder and no longer on the complete archive. The retention information is merely used by the internal process that deletes old e-mails from the archive.

Finally, the original content has to be persisted by the EADM system. Of course this has to be done in an immutable way. Yet it does not have to be searchable, because this is done using the extracted text mentioned above. It only has to be retrievable. That means for a given identifier the system must be able to retrieve the associated content.

In order to reduce the storage consumption of the EADM system we implemented single-instance storage on the part level. To this end the e-mail is separated into multiple parts, where each body and each attachment is one separate part. As the same file is often sent as an attachment in many e-mails, this can lead to a significant reduction of the storage consumption.

Up to now we discussed some properties of the data model, but we still did not show the way to implement it. We now show one approach for mapping the data model onto existing technology as depicted in figure 5.5.

On the left hand side of the figure you can see the storage, where the documents are stored in a way that allows restoring them in their original form. As described earlier the e-mails are separated into a core part, the document, and several other parts. In the e-mail case all the bodies and attachments are stored as separate parts in order to reduce storage consumption. After separating these parts from the e-mail remain the headers of the e-mail, the structure of the parts, and metadata about the parts like the filename of the attachment. This information is persistent as the separate document.

The documents are not stored in their original form, but in a form that allows to reconstruct the original. The disadvantage of this approach is, that when storing an e-mail or when retrieving a complete e-mail, more disk requests are necessary. And there are more and smaller units to be managed. The main advantage is that a part that is common to multiple documents only has to be stored once. It is common to display only the headers and the body of the e-mail in a first step. The attachments are

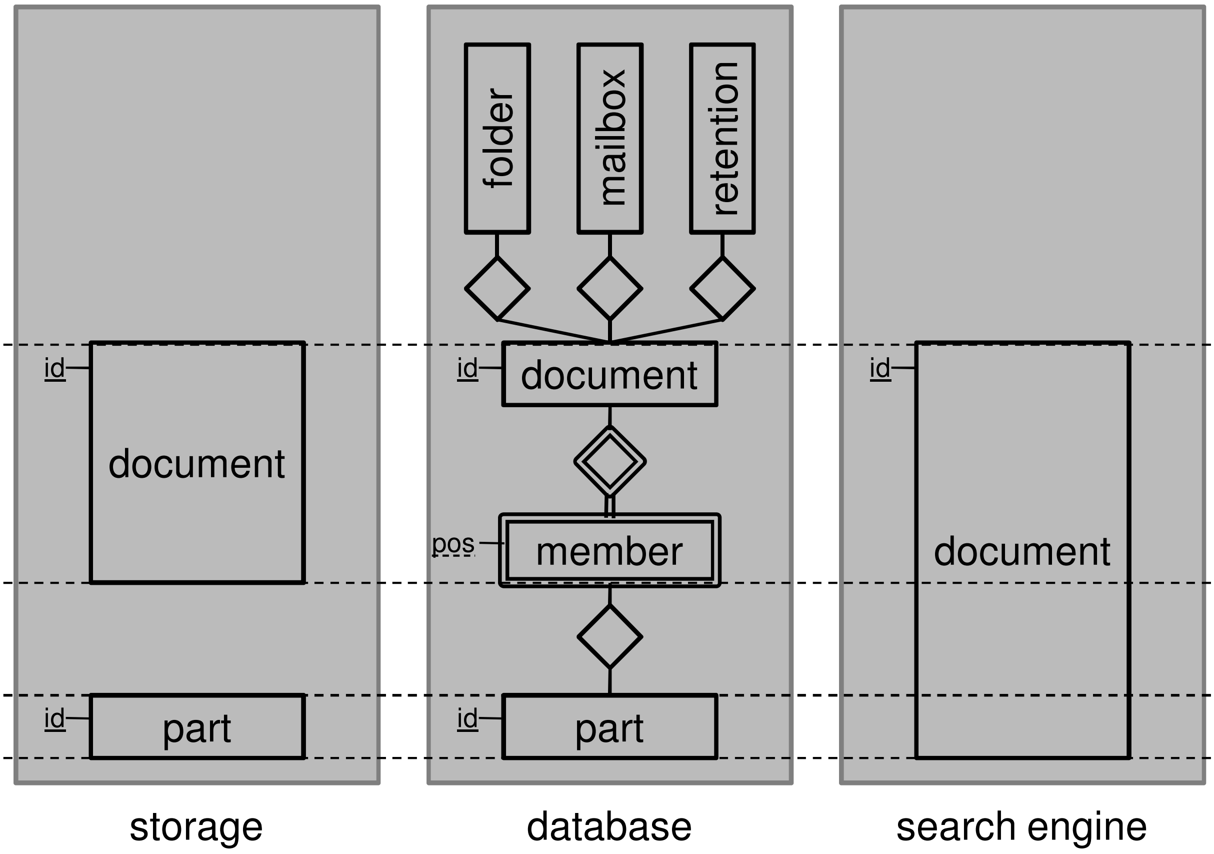


Figure 5.5: Implementation of the data model using different technologies

only displayed when the user explicitly requests them. So being able to retrieve the parts independently can be an advantage.

In the simple case the documents and the parts are stored in files in a file system. The name of the file is computed from the identifier of the document or part. Given the identifier the file is created in a unique location, and it can be easily retrieved from there later on. As the identifier is computed as a hash over the content of the part or the document, the content determines the identifier which in turn determines the location of the file where the content is stored.

Currently only parts in the sense of the MIME standard are treated as separate parts. In practice it can also be beneficial to treat other entities as separate parts. One example are the recipients-fields in the header of the e-mail. E.g. in circular letters to a department or a larger organizational unit these fields can easily become larger than the body of the e-mail. Beyond some size the overhead for the management of the additional part can pay off. At first it is only reused by all the archived instances of the circular letter from the mailboxes of the recipients. When the frequency of circular letters is higher than the frequency of changes to the recipients of the distribution list, this separated field is also reused by different letters.

In practice it can be beneficial not to manage all parts separately. Some parts are so small that the stub that links to the separately stored part is hardly smaller than the part itself. In such a case the overhead doesn't pay off. To avoid such an overhead, a configurable minimum size for parts that are managed separately can be added easily.

In the center of figure 5.5 the relational database is shown. It is mainly used to store the mutable information and the structure of the content. The retention category determines the point in time when a document can be deleted. The folder information is mainly used by the compliance officer when working with his cases. But assigning a document to a case implicitly sets this document on hold. Thus the folder information is also required when deleting documents. The relation between the documents and the parts is required to determine the point in time when a part can be deleted. This is only allowed when it is no longer referenced by any document. Therefore it implicitly depends on the retention category of the documents that reference it and on their on-hold status.

Documents are added to a folder by saving the hits of a query. This is done by the compliance officer when searching for documents that are relevant to some case. In a first step he tries to build a query that retrieves the relevant documents. He executes this query and inspects the top hits. When he is confident with the result he can execute the query and persist the complete result set into a folder. After that he does no longer working on the complete archive, but only on the folder. He can look at individual documents in the folder and also remove them from the folder. Finally he typically exports the documents so that other persons can work with them.

The name of the owner of a mailbox is required when processing most search queries. For typical end user queries it is necessary to enforce access control restrictions. The user should only get hits for documents he is allowed to see. The compliance officer is allowed to see all e-mails, but he can restrict the set of mailboxes for his search to a specific set of users.

The search engine on the right hand side of the figure stores the extracted plain text from the headers of the e-mail, the body and the attachments in an inverted index. In addition, some fields are also stored in the search engine. These fields are used when displaying the hits for a query to the user.

In the previous section about search engines we mentioned that search engines do not support updating parts of a documents and joins between different data types in the index. One solution to get around with these limitations is denormalization. In the case discussed here each document in the index represents one complete e-mail with all its attachments. So each occurrence of an attachment in an e-mail consumes space in the index. Typically this is only around 10% to 20% of the original size of the attachment, but still it consumes space. Further, during the ingestion all attachments have to be processed to e.g. extract the plain text. It is not possible to skip already known attachments as the extracted text is needed to build the entry in the search engine. In section 5.1.8 we evaluate another approach where the parts are entries of their own in the search engine and are thus inserted only once.

When ingesting a new document into the archive, all three systems have to be updated. The content in its original form is stored in the file system, an entry for each document and each e-mail as well as for each relation between them is made in the database, and the extracted plain text is added to the search engine. The strict and consistent way to do all these updates would be to perform them in the scope of one global transaction. This would ensure, that either all updates or no updates at all are persisted. The downside is a high coordination overhead to synchronize all the updates.

As shown in section 5.1.5 an EADM system does not require such a strict consistency. Especially it is not necessary that new updates are visible immediately. A window of a few hours is tolerable before the system has to be consistent. This can be seen as a form of eventual consistency [Vog08].

An additional precondition here is that while a document is processed it is still reliably stored at the source of the documents, in the concrete case on the e-mail server. Thus in case of an error the documents can be re-read from the source. Only when the whole ingestion process was successful the document is modified at the source.

Using a content-based identifier as the key for parts and e-mails has an important consequence for the consistency of the ingest. When the ingestion of a document failed for some reason, it will be retried later. The content-based identifiers make sure that in each attempt the identifiers are the same. Thus when repeating an ingestion it is possible to check whether some parts of the ingest were already successful.

To deal with failed jobs it is necessary for this method that the individual operations during the ingest are idempotent. It must be possible to repeatedly apply them with always the same result. For the three systems considered here this is realized as follows:

- When storing the original data in a file the identifier is used to compute the name of this file. If a file already exists a store request can simply be ignored as it would only overwrite the file with the same content.

- The database ensures the uniqueness of the data by enforcing unique keys. In case of a repeated operation with the same data the database will raise an error and the application only has to deal with it appropriately.

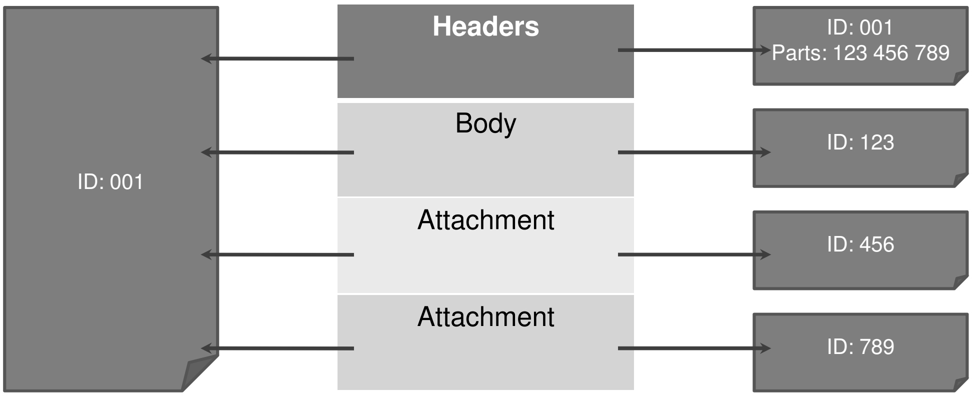


Figure 5.6: Denormalized (left) and normalized (right) mapping of an e-mail to documents in the index

- The search engine does not have the knowledge about application-specific unique fields and thus does not prevent the repeated addition of a document. Additionally the loosened consistency model does not allow querying for recently added documents. Thus another mechanism is needed here. One option is to persist the changes within the search engine only at the end of a job. In this case no information from a partially failed operation ends up in the index. Another option is to tolerate duplicates in the index in case of errors and to filter out duplicates when processing a query. As the duplicates have the same score they should be nearby in the hit list.

5.1.8 Normalization in the index

In the previous section we implemented a denormalized data model in the index where each search result is the result of one document in the index. This is the normal way to use a search engine and typically results in a good response time.

When looking at the data an EADM system has to deal with, the e-mails that are sent within and between enterprises, one finds a lot of repeated data: 1) some attachments are attached to different e-mails, and 2) there are often multiple recipients for one e-mail. Especially the second point has a high potential of saving storage space as the same e-mail is in the mailbox of each sender and recipient.

This large amount of repeated information was the motivation to investigate normalization within the index further. The basic idea is to index the parts separately in order to reduce the size of the index and the effort necessary to add the parts. This approach is depicted in figure 5.6. In the center of the figure is the e-mail with several headers, one body, and two attachments.

On the left side you can see the previous approach where all the information from the e-mail is added to one document for the index with several fields. Of course there are several more fields, especially in the header, but the figure only shows the information relevant to distinguish the two options.

The normalized representation is on the right side of the figure. In this case each part, body or attachment, is represented by a separate document in the index. In addition to the text extracted from the part this document has a separate ID generated from its content. An additional document in the index represents the e-mail and contains the information from the header of the e-mail. A further field contains references to all the parts that belong to the e-mail.

Looking at one e-mail only there is no advantage of this approach. But with many e-mails it is possible, that multiple e-mails reference the same part. In this case the part only has to be indexed once. This does not only reduce the size of the index, but it also allows skipping text extraction for not indexed parts.

The information whether a part is a duplicate or not is gathered from the database. It is available as a by-product when updating the parts table in the database. A flag will be set for each duplicate part such that subsequent steps can be pruned. In case of a failure it is possible that the index and the database are not in sync. In that case a job has to be repeated in a special mode where the index must also be queried to determine duplicates.

The more complex data model requires a more complex processing when executing the queries. One problem is that hits within a part will not deliver the necessary information about the corresponding e-mails. Thus an additional query to look up the e-mails that reference the part has to be performed. These queries can be batched to reduce the number of queries sent to the search engine, but nevertheless this is an overhead.

Another problem is that there can be one hit in the headers and another hit in a part of an e-mail, or there can be hits in different parts that are referenced by one e-mail. All these hits for one e-mail have to be combined in order to calculate the final ranking of the e-mail. In the current implementation this is realized by a hash join. Only after computing this join the results are displayed to the user.

5.1.9 Test Environment

Before presenting the results of the performance studies we briefly describe the test environment we used. This includes the hardware, the tools, the data and a general description of the tests. This environment is also used in further sections.

The test system mainly consisted of an IBM BladeCenter with HS20 blades with two 3,2 GHz Intel Xeon CPUs each, and HS21 blades with two dual-core 2,3 GHz each. The Linux distribution CentOS version 4.7 runs on theses blades. In addition there are two JS21 blades with two PowerPC CPUs each running at 2,7 GHz. One of them is used as the e-mail server with the operating system AIX 5.3 and the e-mail server Lotus Domino 7.0.2. All blades have 4 GB of main memory, a 73 GB hard disk and are connected with two 1 GBit-Ethernet switches.

An IBM DS4100 storage system is attached to the blades using fiber channel. It contains eight 400GB SATA disks that form one RAID 5 array. Different logical units are defined to hold a cluster file system, to provide an additional data partition for each node and for the mailboxes of the e-mail server.

Three different data sets were used for the tests:

random domino/large: The body of these e-mails was generated as a sequence of random words from a dictionary. Additionally an attachment was added to 30% of the e-mails. 50% of the e-mails were sent to one recipient, and 50% had two recipients. The mailboxes of these recipients are managed by a Domino server on another blade. As can be seen in table 5.4 these e-mails are larger than the other e-mails, and especially they create a lot of tokens during text extraction.

enron domino: This data set is based on the enron data set that was made public during the investigation of the enron scandal (see e.g. [ME04]). During the export the attachments were removed from all the e-mails. This is disadvantageous for our case as the processing of the attachments by the archive is typically a major and costly step. To circumvent this deficiency we measured the distribution of the attachments for a real mailbox and then randomly added documents crawled from the Internet to the enron e-mails. The appended document set contains PDF and HTML documents as well as Microsoft and OpenOffice documents, but also images and binary data. The average size of these e-mails is still smaller than the random e-mails, and they also have attachments that do not contain text like images. These mailboxes are also managed by the Domino server.

enron zip: The mailboxes were created similarly to the previous one. But in this case the mailboxes were stored as MIME messages within a ZIP file, where each ZIP file contains the e-mail for one mailbox. The ZIP files were stored on a cluster file system on the DS4100 so that they are available to all the nodes. The main reason for this configuration is to have a data source with adequate throughput without having to set up a sufficiently powerful e-mail server infrastructure.

	random large	**random domino**	**enron domino**	**enron zip**
mailboxes	99	30	150	150
e-mails	1006794	99916	517430	517431
distinct e-mails	651920	66666	517430	517431
distinct parts	692282	79325	384988	259423
average size [bytes]	153566	152456	31993	40660

Table 5.4: Different metrics about the test data sets

Some metrics about the four data sets are given in table 5.4.

For multi-user performance tests we use JMeter[9] from Apache. JMeter is a versatile open-source tool to measure the performance of a system. JMeter can test many different actions. In our case we are using it to submit HTTP requests to our user interface that trigger different actions in the EADM system. In the tests these are different search requests for e-mails.

The test plan used for the search tests was originally defined in [Sch06] and recently modified in [Gon09]. At the beginning of each test a set of concurrent threads is started. Each thread simulates one concurrent user. The thread repeatedly sends a request to the system, receives the response, and then waits for a think time between one and five seconds. In average three seconds is a comparatively short time, and reflects a worst case usage scenario. When this time is over the user starts from the beginning and issues another request.

The test plan contains 60 different queries grouped into 6 categories with 10 queries per category. The categories and their relative frequencies are:

- Single term queries: 30%
- Multi term queries: 30%
- Phrase queries: 25%
- Boolean queries: 7%
- Wildcard queries: 5%
- Fuzzy queries: 3%

For most tests a restriction was made by disabling the wildcard queries and the fuzzy queries. With a large index they create too many results. Thus the results are often not

[9] http://jakarta.apache.org/jmeter

very useful, and processing them requires a lot of resources. In this case the percentages of the remaining query types were proportionally increased to add up to 100%.

Further constraints are randomly added to the queries. One constraint restricts the query to one or a few mailboxes. Another constraint restricts the query to only contain hits from a specific date range. The frequency with which these are added is given in the following table:

	date restriction	**no date restriction**
with mailbox restriction	60%	15%
no mailbox restriction	20%	5%

The version of Apache Lucene we are using in the tests is 2.2.0. This is a slightly outdated version, but it showed a better performance in our scenario than the 2.4.0 version. The web services and the user interface are running in an Apache Tomcat server version 5.5.20 using the Java 1.5.0 virtual machine from IBM with build level 2.3. The separate processes like the search server are using the Java 1.6.0 virtual machine from IBM with build level 2.4. Both JVMs are 32-bit editions as this is required by the text extraction library we are using.

5.1.10 Catalog Performance

The tests in this section were performed on one of the HS21-blades. The steps that were performed for each e-mail are hash calculation, catalog database update, text extraction and indexing. The crawler, the index and the catalog database were running on the same node. The database, the index and temporary files were stored on the local disk. The log of the database was kept on an in-memory file system in order to reduce the problem with the disk-performance-bounded system. In a practical scenario it should be placed on a separate disk.

Figure 5.7 shows the average time to process an e-mail for the different data sets and for a different number of threads in the crawler. The red line shows the total time to process the e-mail, and the stacked bar shows the time consumed in the different processing steps. The processing steps do not include opening the connection to the data source and fetching the references to the e-mails. This explains the gap between the line representing the total time and the stacked bar representing the time in the processing steps that is visible in the single-threaded case. Conversely this sum is higher than the total time with multiple threads as in an extreme case each thread could work for the total time.

In the graph the time needed to calculate the content-based hash for the e-mail and its parts is relatively high. This is largely due to the fact that the hash calculation is the

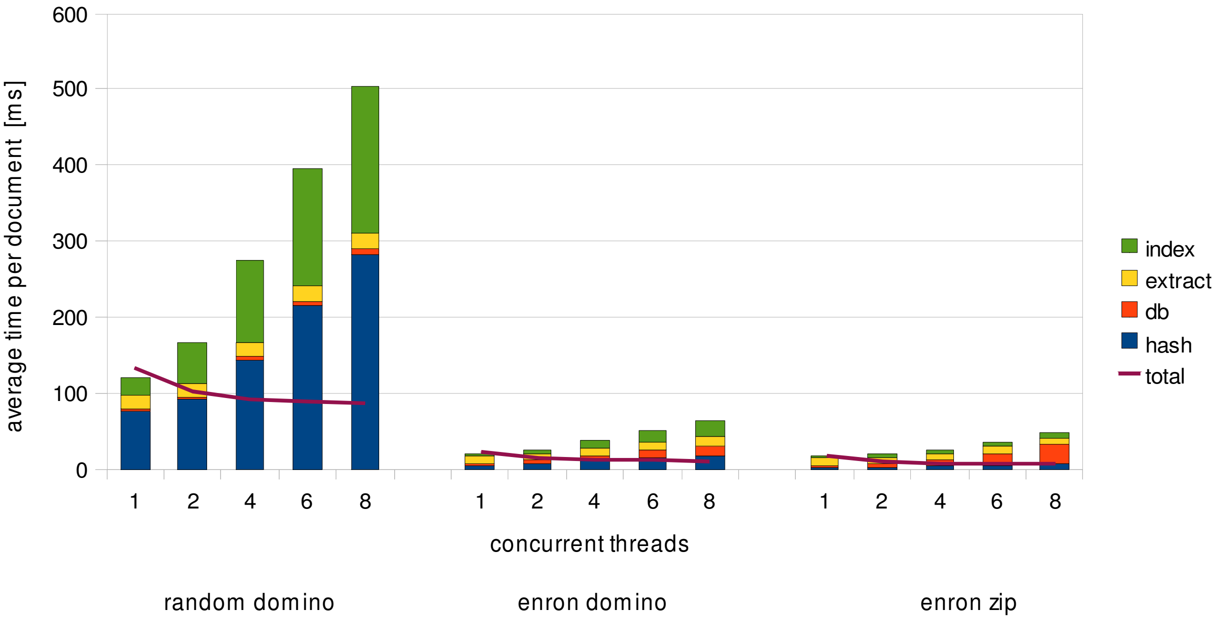


Figure 5.7: Average time to catalog a document for different data sets and different numbers of threads

first step when processing each e-mail. Thus the time needed to fetch the content of the e-mail from the sources is attributed to the hash calculation. The other subsequent steps simply reuse this content without fetching it again.

What can be seen from line representing the total time in the graph is that increasing the number of threads increases the throughput to some extent. But the benefits are not as huge as possible. The reason therefore is that for some of the steps the time per document clearly increases with the number of threads. With the large randomly generated e-mails on the domino server, fetching the e-mails from the source is a problem as can be seen from the large fraction of the hash calculation. It seems that the connection to the Domino Server does not support concurrent operations. Also the large amount of tokens in each document leads to a synchronization within the indexing. For the relatively small e-mails in the enron zip collection the updates of the catalog database become the biggest problem.

The measured total time (red line in figure 5.7) shows that the performance of the ingest process can be increased by using a larger number of concurrent threads. Yet one soon reaches the boundaries of the different subsystems. Thus with a scale-up approach one has to start tuning the different subsystems. And this is an ongoing task as changing characteristics of the e-mails requires the adaption of parameters. Especially when the available disk-bandwidth of the node with only one disk is really low.

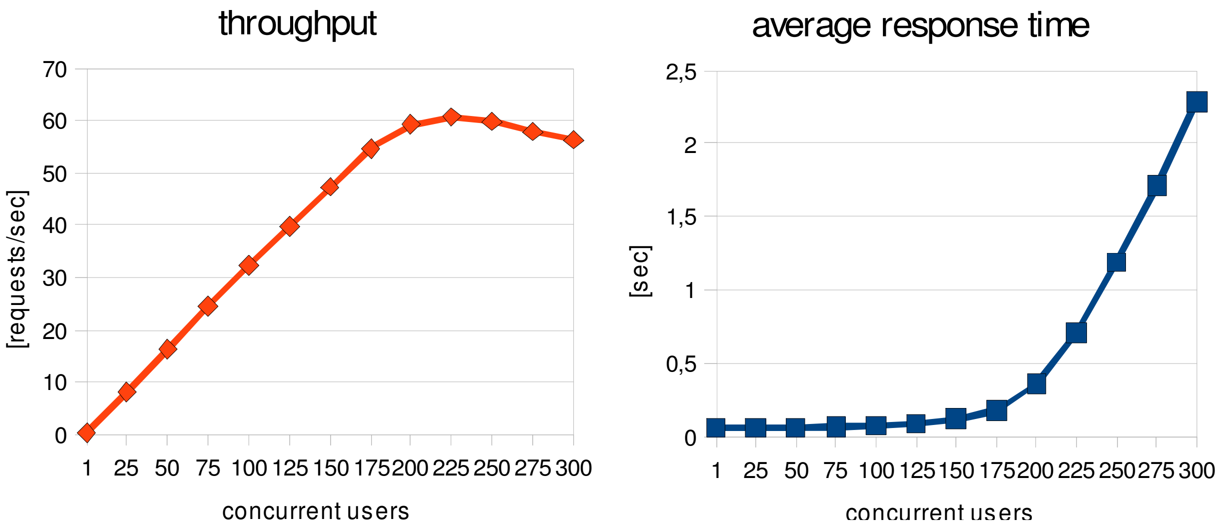


Figure 5.8: Multi-user search performance with one search server

Figure 5.8 shows the search performance with one index on one search server. The index contained the documents from the “random large” data set and was optimized before running the test. The user interface preparing the results for the user ran on a separate machine. The system can maintain an average response time of below 0.26 seconds with up to 225 concurrent users. At that point the CPUs on the node with the index are nearly saturated. With more users the response time rapidly increases and the throughput no longer increases.

In the following we look at some measurements when using deduplicated documents in the index. As already said in a previous section all the parts are indexed separately in this case and each part as identified by its content-based hash is only indexed once.

In figure 5.9 the ingest performance of the two cases with whole documents in the index and with separated and deduplicated parts are compared for different data sets. We performed the same tests as before but kept the number of threads fixed at four threads. In fact the stacked bars for the whole documents in this figure and the stacked bars for four threads in figure 5.7 are based on the same measurement. As can be seen at the line representing the total average time per document the deduplicated indexing process is always faster. With the smaller enron data sets the processing step that benefits most is the text extraction. But also the time for indexing the documents is reduced. For the larger randomly generated e-mails it takes relatively long to retrieve the e-mails from the data source. This time is included in the time to calculate the hash and is even larger compared to indexing whole documents. Yet this is mostly waiting time.

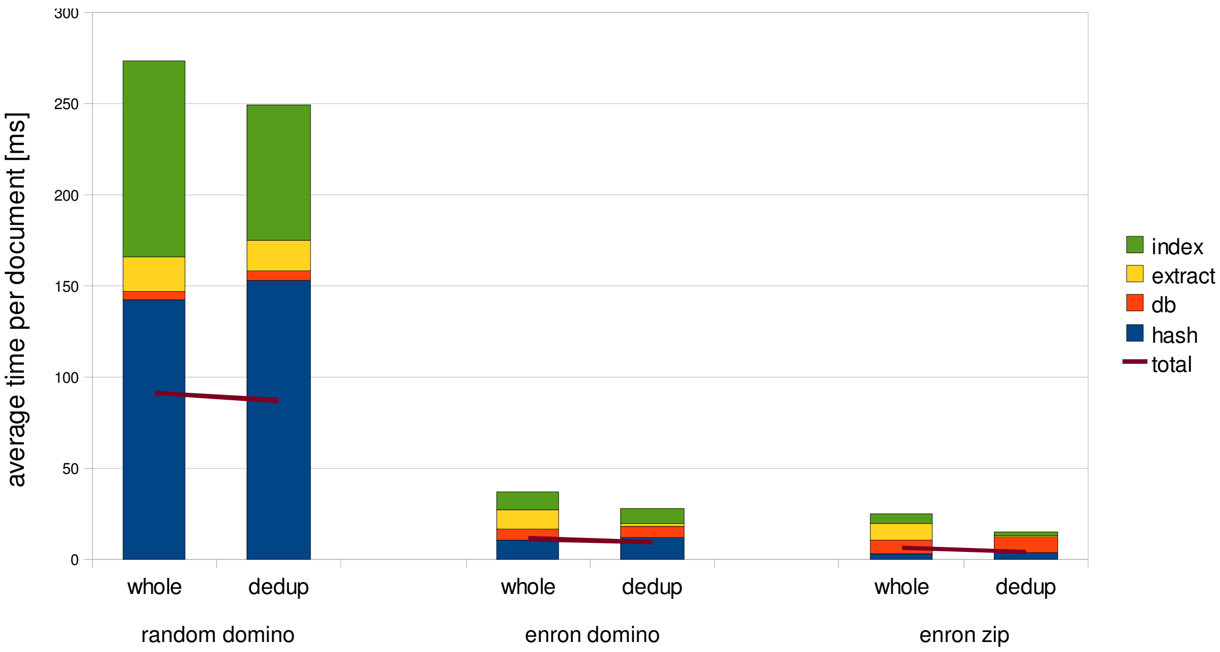


Figure 5.9: Average time to catalog a document for whole-document index and deduplicated index for different data sets

Table 5.5 shows some statistics for whole document indexing and deduplicated indexing for the different data sets. The most interesting fact is that the size of the index is reduced by 28% to 44%. Thus a considerable storage reduction for the index is possible with the deduplicated indexing.

The search performance with the deduplicated index is very bad. Simple queries with only one keyword can take one hundred seconds to complete. With this long execution time it does not take many concurrent threads to exhaust the system. With thirty threads this can already lead to out of memory exceptions.

The query performance is acceptable when the selectivity of the query is high enough and does not create "large" intermediary results. This especially comes true for the part of the query that affects the extracted text from the parts. Whether a query is answered quickly can only be guessed upfront. A time-out mechanism could be used to abort long-running queries. But the system already consumed resources when the query is aborted. As it is not apparent from the query why it is aborted this would be very inconvenient for the user.

The deduplicated indexing was an attempt to reduce the size of the index and processing time necessary to create it. As discussed the size of the index is reduced by about one third, and the ingest time is reduced by 5% to 37%. Yet the search performance makes this approach unusable, especially on large data collections. And this

data set	**random domino**		**enron domino**		**enron zip**	
indexing	**whole**	**dedup.**	**whole**	**dedup.**	**whole**	**dedup.**
db docs	66666	66666	517430	517430	517431	517431
db parts	79325	79325	384988	384988	384990	384990
db members	86774	86774	781738	781814	780441	780441
index docs	99916	145927	517430	902399	517431	773269
index size [KB]	1458112	1052220	986536	637988	1061632	592444

Table 5.5: Statistics about the metadata after whole document indexing and deduplicated indexing

will get even worse in a distributed setting with a higher latency of the communication between the indexes. Therefore we will use whole document indexing in the remainder of this work.

5.2 Distributed Catalog

In the previous section it was shown how to implement the catalog of the EADM using different technologies. In this section we distribute this catalog in a cluster of multiple nodes. First we discuss several ways to split the data and to assign it to the available nodes. In section 5.2.4 we then discuss the advantages and disadvantages of different criteria to partition e-mails. Section 5.2.6 presents the implementation of the prototype and section 5.2.10 shows the results of performance measurements.

5.2.1 Data distribution

The single-node implementation of the EADM system discussed in the previous section has limitations regarding the achievable data volumes and the amount of data it is able to manage. To overcome these limitations we now cover the implementation in a distributed system. The aim is to develop a system that can be scaled-out by adding more resources.

In section 3.2 we depicted different architectures of multiprocessor systems. The main architectures to be investigated here are shared-nothing and shared-disk systems. These systems can grow easily by adding more nodes. But the way how the data is distributed over the nodes has a high impact on the performance of these systems. In this section we discuss several methods to spread the data over a set of nodes. The motivations are: 1) parallelization of the workload and 2) scalability to large data volumes.

5.2.1.1 Data replication

One option to distribute data over multiple nodes is replication. In the simple case all the nodes could maintain a copy of the whole data on their local disks. The incoming requests can then be spread e.g. round robin to the available nodes. This approach can be used when the CPU or the disk-bandwidth is exhausted and more concurrent users have to be supported. Replication is no option when the data set is getting too large for one system, as each node still has to store and process the complete data set. Therefore the single-user response time is not reduced, but it is possible to serve more users as the nodes can process their requests independently of each other.

Replication also plays an important role when the availability of the data is important. This is of course the case for an EADM system, as otherwise there would be no reason to archive the data at all. Yet we don’t take this into account in the practical tests as the disk subsystem is already a limiting factor without storing replica of the data.

Up to now the discussion only considered read-only data. When the data is also updated, a synchronization of the replica on the different nodes is necessary. Many approaches have been discussed in the literature. One simple approach is master-slave replication, where only the one master can be updated, and this master then ensures that all the slaves are updated, too. Obviously this can result in a considerable load on the nodes. Therefore with replication the rate of reading operations should be much higher than the rate of writing operations.

5.2.1.2 Splitting the data

There are many ways to separate data into multiple segments. One distinction is the type of the boundary around the segments. Maintaining fixed boundaries around the segments means more effort during the insertion of new data as it has to be inserted into the right segment. And when updating data it can be necessary to move it from one segment to another. Yet this overhead can pay off when other operations that otherwise would have to access all segments can be restricted to a subset of them.

The simplest option is having no defined boundary. New data can just be inserted into any segment, e.g. the nearest segment or round robin. The downside is that most other operations have to access all segments because they have no information to determine in which segment the relevant data is. This can also influence the insertion of new data when for example unique keys have to be maintained.

Metadata about the data like statistics or bloom filters might be used to reduce the set of segments that are relevant for an operation upfront. Yet their maintenance is not for free. And when the data is arbitrarily distributed over the segments the selectivity tends to be low.

5.2.1.3 Data partitioning

A more stringent form of the previous case is partitioning, where boundaries between the segments are defined and enforced. It is related to the mathematical partitioning of sets. The whole data set is split into multiple disjunct partitions where each data element belongs to exactly one partition. Thus there is no overlap between the different partitions. There are five basic ways to split the data (see e.g. [Dad96]):

Horizontal partitioning With this partitioning strategy the data set is cut horizontally, that is complete rows are assigned to different nodes. In our example all the information belonging to an e-mail would be on one partition, but information for different e-mails might be in different partitions. The advantage is that operations that apply to one e-mail like inserting a new e-mail or deleting an old e-mail can be executed on one partition only. Also queries with predicates on different attributes like "+subject:report +from:mayer" can be evaluated locally. For the distribution of the rows to the different partitions there are two approaches: range partitioning and hash partitioning. With range partitioning one attribute is declared to be the partitioning key and its values are divided into different ranges. Each range is then mapped to one partition. In our example one could e.g. declare the sent-date as the partitioning key and define one partition for each year.

Hash partitioning is a special kind of range partitioning where not one explicit partitioning key is used, instead the partitioning key is computed as a hash value over multiple or all fields of a row. The hash function typically ensures a uniform distribution of the data, but a query often has to be performed on all partitions.

Vertical partitioning When vertical partitioning is applied, the rows are split into distinct parts and those parts are assigned to different partitions. That is one set of attributes is stored in one partition, and another set is stored in a different partition. This is especially useful in cases where the access pattern to the attributes differs. In the e-mail example it might be useful to store the header fields, which have to be displayed in result lists many times, in one partition, and to store the bodies and the parts, which are only displayed afterwards on demand, in another partition. In this way the partition with the frequently used information is smaller and can thus be processed faster.

Derived partitioning Derived partitioning is an extended variant of horizontal partitioning. In this case not one attribute is used directly to determine the partition of an entity, instead a foreign key is traversed and the entity is stored in the same partition as the referenced entity. This can be used to store parts and the entity they belong to in the same partition. In the variant where the parts of the e-mail are modeled as weak entities the parts are good candidates for such a derived partitioning. The probability in this and in many similar cases is high that the application has to join between the two entities frequently. Using derived partitioning this join can be computed efficiently with a collocated join: the join only has to be computed within each partition, and the global result is computed by simply concatenating the local results.

Mixed partitioning The three partitioning strategies can also be applied recursively to already partitioned data. E.g. one could first do a vertical partitioning to separate the attributes depending on their usage pattern, and then partition horizontally on the sent-date or the identifier to get even smaller partitions.

In the area of search engines the two main approaches to partition the inverted index are term partitioning and document partitioning (also known as global partitioning and document partitioning). With term partitioning one global index is built and each node is assigned to one part of this index. E.g. one node hosts the part of the index with terms starting with a to m, and the other node hosts the terms starting with n to z. In the document partitioning case each document is assigned to one node and the index on this node has all the terms of this document. Depending on the scenario and the tests performed one or the other approach is better (see e.g. [MWZBY07, BBYRNZ01]). We focus on document partitioning as dealing with updates is simpler and it can be implemented on top of the search engine without changing it.

Partitioning increases the effort necessary to insert new data and to update existing data. It is not possible to just insert the data into any partition, instead the data has to be inserted into the right partition. This means that all the data first has to be moved across the network to the right partition, inevitably resulting in some amount of overhead for the processing of the content. In case of an update it can be necessary to delete an entry at one node and to create a new entry at another one.

In cases where the global consistency of the data set is not strict and a delay is tolerated between the time of the actual insert of new data and the time when it is accessible it would be possible to insert new data locally into a temporary data set first and to re-distribute the data in this set in a second step to the right partitions. Yet splitting a data set is not always easy and requires touching each entry again. With Apache Lucene for example an option would be to in turn mark the documents not part of a partition temporarily as deleted and then merge the remaining documents into a new index. But this requires one complete scan of the index for each partition which is not very efficient.

Instead of forwarding the documents to the right partition directly, they can be indexed locally into multiple temporary indexes, one for each partition. These indexes could then be forwarded to the destination partition asynchronously, while keeping track of the discrepancy between the current and the “desired” distribution. This approach would increase the content processing throughput during peak load times. It also increases overall efficiency due to the batched re-distribution of index data, as opposed to the pre-indexing redistribution of original content.

5.2.1.4 Sharding

Sharding is a technique that is used by many high-volume Internet sites today. The main idea is to use a cluster of cheap standard hardware and cheap/free software and to distribute the data and load over these “shards”. In contrast to the partitioning approach, where the database manages the partitioning, the databases in this case are completely independent of each other. The integration into one virtual system is done in a layer above the database.

A lookup table maps the entries to the responsible shard. The criteria of choice is often a user identifier or a customer identifier. This identifier is part of the key of each entry. In other words the data is partitioned horizontally by user/customer identifiers. With multiple data types stored in different tables some of the tables are stored in a denormalized form using a derived partitioning.

Sharding breaks with the basic idea of database management systems to separate the application into the application logic part and the data management part. The application issues its queries in a descriptive query language without having to respect the implementation. And the DBMS has a lot of freedom on how to obtain the results. Now with sharding an important decision is already made before the data and the queries actually reach the DBMS. The downsides of this approach can be increased complexity of the overall system, increased administration overhead and decreased optimization potential within the DBMS.

The positive aspects are that sharding can be implemented on top of any data management system. The responsibility how the data is distributed to the underlying systems is completely within the sharding system and is transparent to the underlying systems. For the same reason the scalability of the underlying systems is not crucial for the overall system. As long as it is possible to instantiate multiple independent underlying systems it depends on the sharding system whether the system can scale-out.

5.2.2 Allocation

The next step after partitioning the data is the allocation of the partitions to the real resources (nodes and disks). Thereby two aspects have to be taken into account: efficiency and availability.

The mapping of partitions to resources does not have to be a one-to-one mapping. In cases with more defined partitions than available resources it is possible to assign multiple partitions to one resource. One possible strategy is to create deliberately more partitions than initially required. The advantage of this strategy is that it is often easier to reallocate a partition to another resource than to repartition the data by splitting and merging existing partitions. Thus it requires less effort to adjust the system to the current load or to re-balance the system in case of a data or load skew. The partitioning sets the maximum degree of distribution, and the allocation allows an adaptation to the current situation.

It is also possible to assign a partition to multiple resources which means replication on the partition level. The system then has to keep the replica in sync.

The allocation strategy can have a huge impact on the characteristics of the system. Chained declustering [HD90] organizes the partitions and replica in a way so that even in the event of a node failure the workload can be equally balanced among the remaining nodes. Adaptive overlapped declustering [WY05] builds on top of this and does not only balance the load between the nodes but also the amount of data they have to store.

Thus having partitions with a fine granularity allows adjusting the system to the current situation better. Reassigning a partition from one node is relatively simple in contrast to splitting a partition. Yet too many unused partitions can also have a negative effect on the performance of the system. In [Sch06] we did some tests with up to 64 indexes on one machine. Except for the wildcard and fuzzy queries the performance was comparable to one single index with the same documents. But these were single-user tests, where in the multi-index case parallelization allows to exploit otherwise unused processing resources. With multiple concurrent users these resources are not unused and thus the result may differ.

5.2.3 Criteria for a good partitioning

In this section we discuss several aspects that stand for a "good" partitioning. At first the partitioning determines which part of the data is managed by each node, and thus the volume of the data. As a consequence this also has a large influence on the load of the different nodes. Obviously the quality of the partitioning largely depends on the workload generated by the application: Insert operations, update operations and delete operations have to be directed to the right partition, and queries have to combine data from the different partitions.

Aspects of a good partitioning were analyzed in [WK83] and the concept of "local sufficiency" was introduced. A partitioning is said to be locally sufficient when each query in the workload can be separated into multiple sub-queries and each sub-query only works on data within one partition. Data only has to be moved between partitions at the end of the query execution to compute the final result as a union of the results from the partitions. Stonebraker [Sto86] calls an application that can be partitioned in this way "delightful" and he asserts that "most database applications are nearly delightful".

With a locally sufficient partitioning, the interaction between the partitions is low and thus a high degree of parallelization is possible before the synchronization mechanism or the communication system becomes the bottleneck. Yet for most operations each partition contributes a part of the result and thus has to be consulted. To be able to prune some partitions from the execution of a query, it must be possible to determine upfront that they don't yield any results. As an example, assume a data set is partitioned by a date value and the query has to retrieve the data for a specific year. In this case all partitions with a date that is not within this year can be pruned. Yet this is only possible when some part of the query correlates to the partitioning criteria.

Another important aspect of the partitioning strategy is how uniform the data is shared between the partitions, and which consequences result on the load of each node. Ideally they would be distributed uniformly over the nodes in order to avoid that individual nodes become a bottleneck. Especially application specific partitioning criteria, like date ranges or user names, tend to be non-uniform. In such cases mechanisms are required to re-balance the partitions. Partitioning by a content-based hash identifier produces in principle a uniform distribution, but pruning queries is rarely possible in this case.

It is nearly impossible to predict the incoming data correctly and thus how the system evolves. Marginal differences may sum up to a large skew, or unexpected changes may modify the data to a large extend. Over the time it might even be desirable to have a skewed partitioning in order to fully exploit replaced or new machines that are more powerful. For these reasons, perpetual asynchronous data redistribution is a necessity when an effective partition structure has to be maintained. This functionality can be implemented as a background service that uses idle system resources to adjust the global distribution and move portions of the data between nodes whenever a significant shift in resource utilization or data volume is detected. Redistribution is also necessary whenever the system is expanded with additional nodes or when portions of the repository are purged.

5.2.4 Partitioning criteria for e-mails

In the previous sections we discussed several partitioning strategies and desirable properties in general. In this section we look at several concrete criteria to partition the data in an EADM system horizontally. The partitioning keys must be unique values in order to determine the unique partition where to store the e-mail. Attributes like the "to"-header with the recipients are not usable because they can have multiple values.

The main differences between the presented partitioning criteria lies in the uniformity of the distribution and in the exploitability during ingest, search or management tasks. Exploitability here depends on whether an operation has to be processed on all partitions or whether it is possible to restrict it to one or only a few partitions. When some partitions aren't needed or affected by an operation, and this can be decided upfront based on the operation and on the partitioning, this can be used to reduce the overall load on the system.

For an e-mail archive there are several promising criteria:

Sender: When a normal user looks for an old e-mail, a selection criteria often is the sender of the e-mail. Thus with partitioning on sender, many queries can be evaluated by only a subset of the partitions.

Mailbox: Partitioning by mailbox can be interesting especially in situations where mailbox owner specific criteria have to be maintained by the archive. Particular for this criteria is that the partition can be determined before opening the mailbox, and retrieving e-mails. Thus during the ingestion process, a partitioning by mailbox can be enforced by the scheduler by an adequate assignment of mailboxes to indexes. When searching the index, the queries of a user are always implicitly restricted so that only hits from those mailboxes are produced that the user is allowed to see. Therefore partitioning by mailbox is almost always exploitable during search.

Date: Especially when the archive contains documents of a long period of time, this criterion is very promising. In most cases users are interested in recent documents. To enforce this, specifying a default date range on the query form should be effective. When necessary a user can still change the date range. Partitioning by date also allows using a tiered approach where recent content is stored on more and/or faster hardware than older content. The big disadvantage is that there is one big hot spot for the ingest. Yet, especially when using the sent-date of the e-mail, the other (previous and future) partitions are updated due to the big variety of this value in reality too.

Document ID: The greatest advantage of this partitioning strategy is the uniform distribution of the documents to the partitions when the ID is calculated as a content-based hash. And it alleviates single-instance storage as it allows to identify duplicate documents. Yet for queries it can only be exploited when the internally generated id is known upfront, which is only the case when retrieving a document explicitly.

Crawler: This criteria is not really partitioning, or at least not partitioning on an attribute of the application domain. The crawler just adds all new documents to one local index. This of course is the best case for ingest. But the downside is that with this strategy all other operations have to be processed on all partitions. Yet,

criteria	search	ingest	delete	uniformness
sender	+	0	-	-
mailbox	+ +	0	-	-
date	+	-	+	- -
document id	-	0	-	+ +
crawler	-	+ +	0	0

Table 5.6: Rating different partitioning criteria for e-mails

with a proper assignment of the jobs to the crawlers it is e.g. possible to realize a partitioning by mailbox.

In table 5.6 the different partitioning criteria are rated with respect to the three operations search, ingest and delete, and to the uniformness of the data distribution. Regarding the search operation, those partitioning criteria have an advantage that can be used to prune the number of partitions to search in. The mailbox has the highest rating as it is expected to occur in most queries. For the ingest operation the rating is lower when the overhead to insert a new document is expected to be higher. Thus one index per crawler is clearly the best. Date is rated lower because one main partition has to carry most of the load. But still there are outliers in other partitions. For the delete operation, it is favorable when all the documents to be deleted are grouped together. Thus grouping by date has an advantage in this case. As it might be possible to determine the date when the document was inserted from the id of the crawler the crawler is rated here a bit higher than the remaining criteria.

In the last column the different partitioning criteria are rated according to the uniformity of the data distribution over the partitions. The document identifiers computed as hash values over the content generate a uniform distribution without additional effort. The distribution of the data in the crawler case can be easily controlled by the scheduler. The other three criteria require the modification of the partitioning followed by a reallocation of the data to change the amount of data in each partition. The date partitioning is rated lowest because 1) with the growing e-mail traffic the new partitions are bigger, and 2) regarding the insert operation only one or maybe a few new partitions will receive all the new documents.

Other partitioning criteria may be desirable or even necessary to realize business requirements in specific situations. For example to store e-mails with the confidential flag set internally, and to use an external service provider to store all other e-mails. Or to make a distinction based on the affiliation of the mailbox owner or his job responsibilities. Specific e-mail treatment based on such criteria could even be enforced by company policy or legislative regulations.

It is also possible to implement multiple levels of partitioning. One example is to partition the e-mails first by the year or month they were sent, and in a second step to partition them by mailbox. Or vice versa. The effectiveness of multi-level partitioning is even more dependent on the concrete distribution of the e-mails than the single-level partitioning. Multi-level partitioning is shown here only as a possibility but the main focus of this work is on single-level partitioning.

5.2.5 Partitioning of stored artifacts

To store and retrieve some content, e.g. an e-mail, an identifier is needed. One criterion is whether the identifier of the content is sufficient or if additional information about the location is necessary. We realized two approaches that are discussed below. In the first approach a segment identifier is required in addition to the document identifier, and in the second approach the document identifier is sufficient.

5.2.5.1 Pre-defined segments

One option is to divide the storage into multiple, pre-defined segments. For example there could be one segment on each server that provides storage to the EADM system. The segments can be managed in a central database. This database keeps track of the defined segments and their current location. This central database has the advantage that there is only one location that has the definitive information and one does not have to search for it. The downside is that this database may become a bottleneck.

To manage the segments with the archived content the tables that are introduced in section 5.3.2 can be used. They are used to manage the different types of storage devices. Yet the tables for storage segments and storage locations can also be used to manage the distribution of the data.

When storing new documents, the scheduler determines the storage segment and location where the documents should be stored. The ingest service then uses the location information to store the documents in this segment. Along with the identifier of the document one also has to keep track of the identifier of the segment in order to retrieve the document later. Otherwise it would be necessary to search for it in all segments. The segment identifier is for example stored as an additional field in the index. To retrieve a document first the segment identifier is used to look up the current location information of the segment in the database, and then the document is retrieved from this location using its document identifier.

When adding a new node to the EADM system a new segment for this node has to be defined in the database together with the location information that is required to access it. After this the scheduler can select the new node when scheduling jobs. Before removing a node the segments located at this node have to be moved to another node unless they are already accessible from other nodes like e.g. in a cluster file system. Moving whole segments from one node to another is possible and besides the actual data transfer only the location information in the database has to be updated. Splitting a segment into multiple parts to, e.g., partially move the data to another node would require an update of the segment identifier for some of the documents wherever it is stored. Such an update would be very inefficient and thus splitting a segment into multiple smaller segments is not supported. Therefore the segment size has to be chosen reasonably upfront.

Above we only mentioned the documents that are stored in the system. Yet the metadata in the database as well as the index segments have to be arranged accordingly.

5.2.5.2 Range-partitioning by document identifiers

The previous approach clearly has its limitations when the arrangement and especially the rearrangement of the data within the system are necessary. It only allows dealing with whole segments at a time. For a dynamic and scalable system a more flexible mechanism to adjust the system is desirable.

The second approach we analyze is a range-partitioning of the document set by the identifier of the document. For a given document identifier, the partitioning defines where the document has to be stored. The advantage of this approach is that besides the document identifier no other information like the segment identifier in the previous case is required to retrieve a document. Therefore it is not necessary to keep a document in the same segment for all times until it is deleted.

A range of the document identifiers is assigned to each node within the system, and each document within this range is stored on this node. To retrieve a document its document identifier is mapped to one node where the document has to be located. To rearrange the data on the nodes one can split a segment, merge adjacent segments, or move the boundary between two adjacent segments. In all cases the underlying data has to be moved to reflect the new partitioning.

The partitioning does not only apply to the documents, but also to the metadata in the database. In the following we focus on this data.

5.2.6 Partitioning scheme implemented in the prototype

In this section we discuss the partitioning chosen for our prototype. The partitioning is also depicted in figure 5.10.

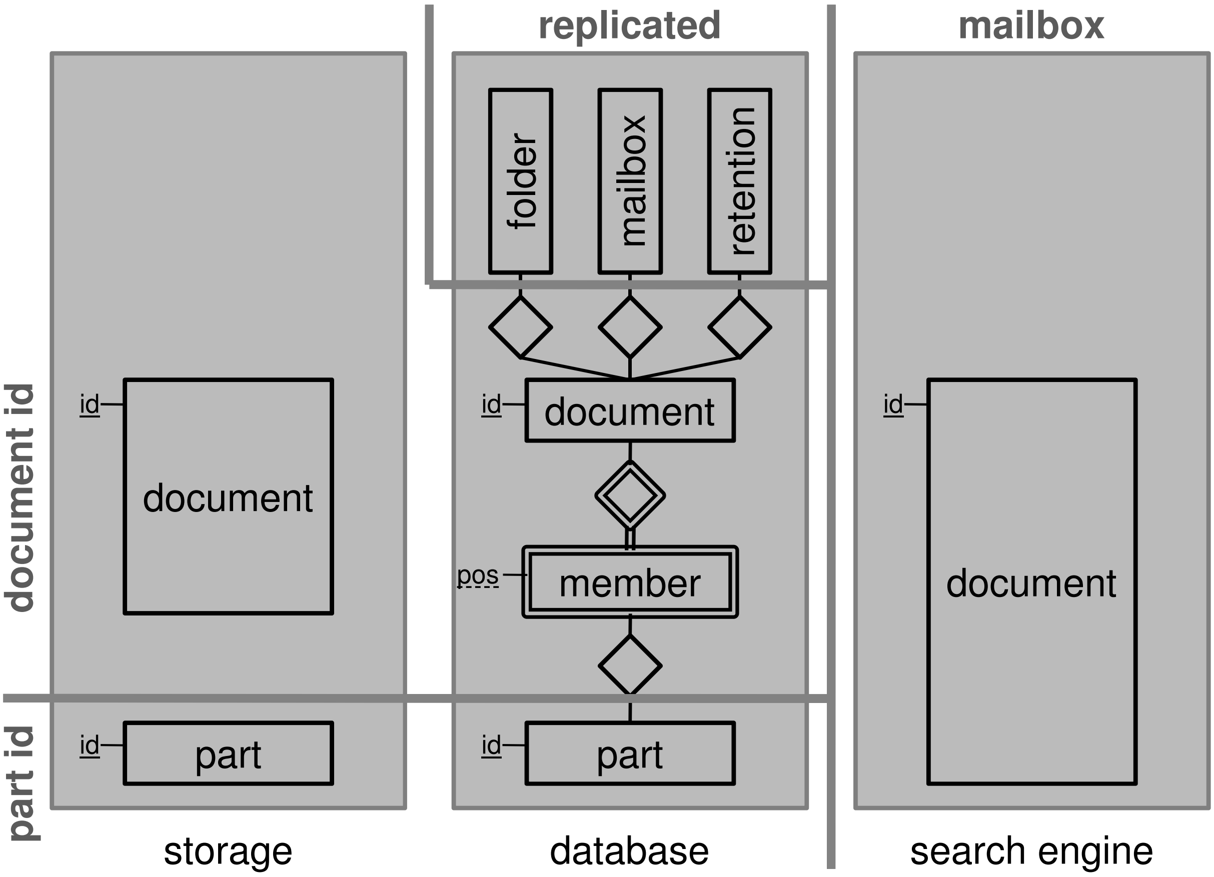


Figure 5.10: Partitioning of the data model in the prototype

For the storage the functional requirements are to store some content under a given identifier, and to retrieve the content associated with an identifier. Thus all operations have the identifier of the content as an input parameter. This makes the identifier an ideal partitioning key. With horizontal partitioning on the identifier all operations can be directed to the one partition that is responsible for it. The documents are partitioned by the identifier of the document, and the parts are partitioned by the identifier of the part.

The database contains more entity types that have to be partitioned than the storage. For the document entities and part entities we also chose to use their identifier as the partitioning key. In fact we will use the same partitioning for the database as for the storage so that the entities in the database and the entities in the storage are collocated on the same node.

The member entity type represents the relation between the documents and the parts. It has two attributes: the id of the document and the id of the part. This n-to-m relationship between the two entity types with the by far largest amount of entities cannot be partitioned in a way that is locally sufficient. Therefore a compromise must be found. As it is more common to search for all the parts belonging to a document than to search for all documents that reference a given part, we chose the id of the document as the partitioning key of the member entity. In this way a collocated join can be used to join documents and members.

For the remaining entity types stored in the database and depicted at the top of figure 5.10 we chose to fully replicate them on all the nodes. Their data volumes and the rate of updates is low enough so that the overhead to store and maintain the replica is tolerable. The advantage is that the information is available and can be directly used at all the nodes.

The search capabilities are provided by a cluster of full-text search engines. When archiving a mailbox according to a job description, the crawler adds all documents to the same search engine. Through an appropriate assignment of the jobs by the scheduler it is possible to create a partitioning by mailbox. This has the big advantage that only one partition has to be opened by the crawler to add new documents. And still this results in usable partitioning criteria.

5.2.7 Implementation using a distributed database

One option to implement a partitioned data model is to use a distributed database like the IBM DB2 Enterprise Server Edition[10] with the Data Partitioning Feature or the Oracle Real Application Cluster[11]. These DBMS promise horizontal scalability of the database while providing the simplicity of a single database to the user.

An obvious idea is to simply use such a distributed DBMS for the catalog with all the metadata of an ECM system. In a project funded by IBM we evaluated this approach using the IBM DB2 Content Manager (CM) in a version shortly before the first shipment on Linux, and the above mentioned edition of IBM DB2 version 8.1.6 also on Linux. Some of the results have been published in [MWM05].

The workload that was used for the evaluation was not EADM, but instead a more general ECM workload. The simulated users created folders, created documents with parts and stored them, moved the documents to the folders, retrieved the documents, moved the documents to another item type and also did some workflow tasks. Thus the workload was more interactive and there were more updates in contrast to the EADM workload. More details about this workload can be found in [Bau03].

The catalog of CM, called library server (LS), was installed on multiple nodes using one distributed LS database. The LS is implemented as a set of stored procedures run-

[10] http://www.ibm.com/software/data/db2/udb/edition-ese.html

[11] http://www.oracle.com/database/rac_home.html

ning within the LS database. Thus by using the distributed database not only the data and the queries are distributed over multiple nodes, but also the application logic. The main focus was on the partitioning of the data, and only small changes were accomplished to the stored procedures.

The key to deliver horizontal scalability for OLTP applications in this environment is to avoid costly communication between the partitions. This is done by collocating all related data in one partition, the relationships being explicitly defined by referential integrity constraints or implicitly by join conditions, and by directing the client application to the partition that contains the data.

The data model of the LS database is highly normalized with many referential integrity constraints. This interwoven set of tables made it very hard to find a suitable partitioning. In fact some constraints enforced by the database schema have to be disabled. One example are different unique keys defined on most tables that can't be enforced at the same time in the partitioned environment. In the case of these keys dropping one constraint is not a big problem, as the generation of the keys is done in a way so that when the one key is unique the other is also unique.

The bigger problem with multiple keys on one table is that there is not only one way to reference or join a table. E.g. one foreign key may point to the one key, and in a join the other key is used. In such a scenario it is not possible to define the partitioning in a way so that the execution of integrity checks or joins is always collocated. Thus only a compromise is possible.

Another problem arises from the strong consistency a DBMS enforces. At the end of each transaction the state of the database has to be consistent again. When the transaction does not change anything this is trivial, but it can also require checks involving the access to many tables. When a row is deleted the DBMS has to check whether references by foreign keys are broken and has to trigger the action specified by the foreign key. This is a big problem with a central "referential integrity" table in the LS, that is referenced from many other tables. When deleting a row in this table the DBMS has to check for key violations in many other tables, and this often resulted in deadlocks. A similar problem occurs when keys are modified. This happens e.g. every time a document advances one step within a workflow.

Although the application that was used in this project was not EADM, and the results were not very satisfying, we nevertheless learned several lessons:

- Simply using a distributed DBMS underneath an existing application is not always that simple. A data model that works well on a single node may have aspects that are not optimal in a distributed environment.

- Keep the data model as simple as possible.

- Consistency is a major problem in a distributed environment. Thus looser consistency models should be chosen when the application does not prohibit them.

5.2.8 Peer-to-peer implementation of the RVL

An alternative to a centralized approach is provided by P2P systems. In such systems many equal peers work together to provide some service. The peers form an overlay network on top of the Internet or the intranet that is responsible for resource location and message routing.

A simple application build on top of many P2P systems is a distributed hash table (DHT). Like a regular hash table a DHT has two main operations: A put operation that stores a given <key,value>-pair in the hash table, and a get operation that retrieves the value that was stored previously under a given key. Yet a DHT is distributed over the peers within the overlay network and each peer is only responsible for a subset of the complete hash table. When accessing such a DHT the request to store or retrieve an entry has to be routed to the right peer first, and this peer then performs the operation on its local data structures.

Many different overlay networks were developed in the recent years. The following discussion is based on a Chord overlay network [SMK$^+$01], namely the OpenChord[12] implementation from the University of Bamberg. Yet, the conclusions of this analysis are, more or less, also valid for other DHT implementations, such as Kademlia [MM02] or Pastry [RD01b]. The first part of this section is about Chord in general, and the second part discusses our additions.

The logically central element of Chord is a ring of large numbers [SMK$^+$01]. The identifier of each node as well as the key of each stored object is mapped to a consistent position on this ring. The partitioning of the objects is now defined by the position of the nodes on this ring. Each node is responsible for all the objects on the ring up to but excluding the previous node. Or seen from the object the object is stored on the first node on the ring after the point where the key of the object is mapped to.

Figure 5.11 shows a simple chord ring. In this example the numbers in this ring are from a very small range only between 0 and 15. The five nodes N1, N6, N9, N13 and N14 are linked to their location on the ring. A node has to store the objects whose identifier is clockwise before them on the ring. Node N1 for example is responsible for objects with the identifiers 15, 0 and 1, and node N6 is responsible for objects with identifiers between 2 and 6. The objects that are actually stored at a node are displayed aside the nodes. For example node N1 stores the objects O15 and O0 and node N6 stores the objects O2, O3, O4 and O6. A new object with identifier O11 would fall

[12] http://open-chord.sourceforge.net

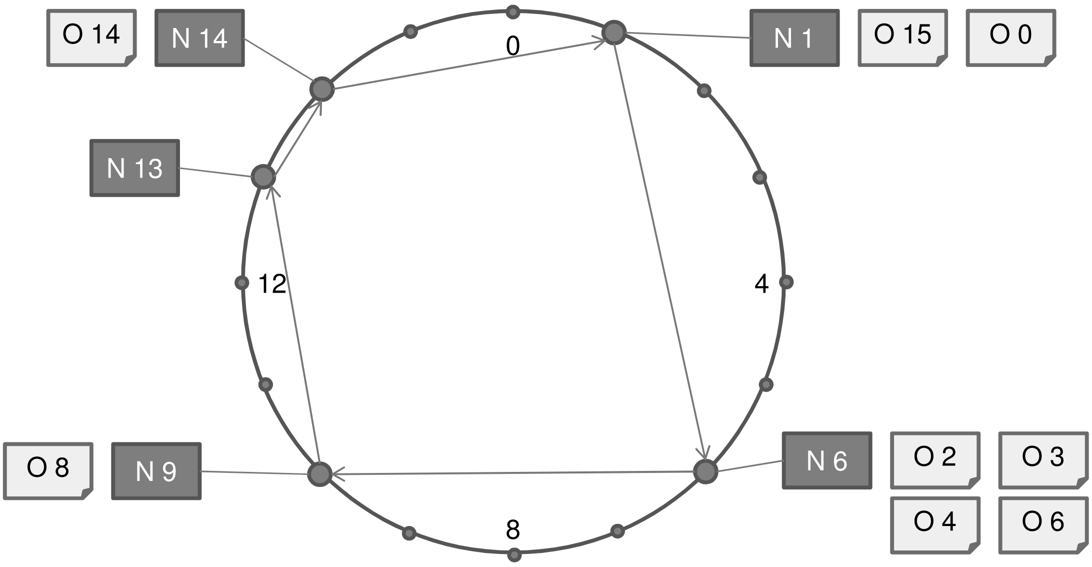


Figure 5.11: Example Chord ring with five nodes and eight objects

between node N9 and N13 and thus node N13 would have to store it.

Looking up an object could be done by walking around the ring from node to node following the successor reference in Chord until the responsible node is found. In average one half of the nodes must be visited so the complexity is within $O(N)$ steps where N is the number of nodes in the ring. But $O(N)$ steps are too much for a large system. The other extreme is to maintain the information about all other peers at each node. This allows to directly communicate with each node, but requires a routing table with $O(N)$ entries. The consistent maintenance of such a routing table is unrealistic in a large system. Therefore Chord has chosen a compromise and in addition to the successor it also maintains a so called finger table at each node with $O(\log N)$ references to other nodes. This is done in a way so that a lookup is possible with only $O(\log N)$ steps.

The EADM system has to perform a lookup using the DHT for every archive, retrieve, insert or update call. Although $O(\log N)$ means only a few hops even for thousands of nodes, i.e. for 16384 nodes about 14 hops, it already needs 3 hops for 8 nodes. This may affect the performance of small or medium sized EADM system negatively. We want to achieve a good scalability by scale-out, but the size of the running system would rarely exceed a couple of hundred nodes. In scenarios of this size, nodes can store routing information of all available nodes in the system, allowing zero hop lookups. $O(1)$ DHT systems, allowing zero hops, are under active research (see [MA06, RS04]) and are used in productive environments, such as in Amazon's Dynamo [DHJ$^+$07] key-value storage.

This cost metric only captures the amount of messages sent. Yet, when the message arrives at the responsible node the identifier has to be looked up in some local data structure. The assumption is, that the message routing step clearly dominates the local lookup. In a typical P2P scenario, where nodes are widely distributed over the Internet and the amount of data per node is quite small, this is a valid assumption.

In our implementation we chose to run the nodes of the DHT in separate Java virtual machines. To access the DHT from the other components we first used a Java RMI wrapper for the DHT nodes. This wrapper was later replaced by a slave DHT node. On startup each slave node selects one of the real nodes in the DHT as its parent. For a lookup of the responsible node for a given identifier the slave forwards the request to its parent. This approach adds one additional hop for each operation. Yet in contrast to the RMI implementation the network traffic was reduced by about two thirds.

Several background tasks of the Chord implementation called stabilization tasks make sure that the routing information stays in sync. They test the reachability of the successor node and they update the finger table as necessary.

As aforesaid, we implemented our own set of operations on top of OpenChord. Most of the operations are processed as follows:

- The responsible node for the given identifier is determined. In this step the DHT performs the above mentioned lookup.
- When the node itself is responsible for the identifier it processes the operation and then returns.
- Otherwise the request is forwarded to the responsible node. The responsible node repeats the same process including the lookup. This is necessary as the structure of the DHT may have changed in the meantime and the node just lost the responsibility for the identifier.

An exception to this process are operations that have to be processed at each node. For example there is a commit operation that has to be invoked after ingesting the e-mails from one job. For such operations we are walking around the ring once and execute it at each node. For a P2P system of the size that we expect for the EADM systems this is a viable solution. More efficient approaches to realize the broadcast can be found in [MG05]. Nevertheless, the aim is to keep the number of such operations at a minimum.

The operations that are implemented mainly fall into two categories: catalog operations and storage operations. The catalog operations are used to forward the queries, inserts and updates of the database part of the catalog to the right node. It thus realizes the partitioning of the database discussed in section 5.2. The storage operations will either serialize the content and send it over the DHT to the responsible node where it is stored in e.g. the file system or in the resource manager, or they retrieve it from there and send the content back to the requester.

The one important operation that is missing is the deletion of the content. With the data scattered over multiple distributed systems this is not trivial. Deleting content one by one would induce a high overhead. An advantage of the EADM application is that the content is not deleted interactively, but it is instead deleted automatically when it is over its retention period and when it is not on hold. This allows organizing the deletion more efficiently. First the records of the documents and the members are deleted from the database. This can be done independently on the different partitions, and all the required information (retention period and hold state) is locally available. While doing this, the identifiers of the documents and the referenced parts are collected. After collecting enough identifiers, they are grouped and sent in batches to the other systems. This will reduce the overhead of the delete processing. Additionally this could be done in times with a lower utilization of the system, e.g. at nights.

In summary, we use the DHT to manage the horizontal partitioning of the data by its identifier and to forward the operations to the responsible node. At the individual nodes we use a different mechanism like the database or other storage back-ends to realize the local execution of the operation.

A disadvantage of a DHT like any other hash table is that it does not efficiently support searching for objects. The only operation it supports is looking up the object associated with a given key. Without the key one cannot get information out of the DHT efficiently. Thus in our EADM system it can be used to retrieve the e-mail or the attachment for a given key, but it is not possible to e.g. search for all the e-mails from a given user. For such operations another mechanism is required. To this end we use a separate cluster of search engines.

5.2.9 Distributed Search Engine

To realize the search functionality we implemented a distributed search engine. The architecture of the implementation is shown in figure 5.12. The node at the top includes the user interface (UI) and a component that executes the search requests of the users. The UI and the distributed search are shown as only one component each, but it is also possible to deploy them on multiple nodes. The distributed search uses a cluster of search servers on multiple nodes to do the actual search.

For the search engine a sharding-approach is used. The inverted index is split into several shards. These shards are assigned to different nodes, where one node can also provide access to multiple shards. One important characteristic for the further usage

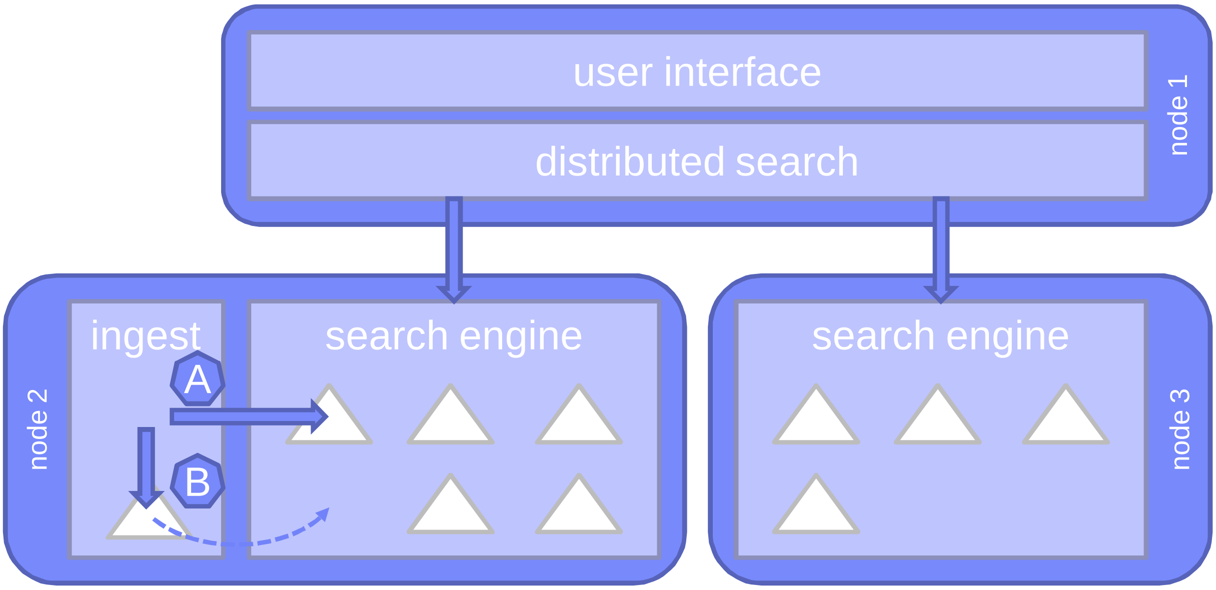


Figure 5.12: Cluster of search engines for full-text search

of the shards is that they can be easily packaged and moved to another node. In the concrete implementation they are only a set of files within one directory.

When processing a search request the UI first has to determine the relevant shards and the set of search servers that currently provide them. This information is retrieved from a database. The request is then forwarded to these search servers. The requests are processed partially in parallel by those search servers and each of them generates a list with its local search results. The UI then receives theses local lists and computes the global result list. Finally it is presented to the user.

When adding new documents to the EADM system the index also has to be updated. There are actually two options, which are also indicated in the figure. With option A the new documents are directly added to index shards that are also used by the search servers. The search engine therefore has to provide a mechanism to synchronize the updates and the search requests. With option B the documents are first added to a separate index, which, when completed, is assigned to a search server.

Several properties are important for how the search engine is operated:

- Newly added documents do not have to be searchable immediately. This allows organizing the newly added documents more efficiently than having to process one transaction per document.
- We allow multiple shards per partition. Thus a document might be in one of several shards. This slightly increases the effort when searching for documents,

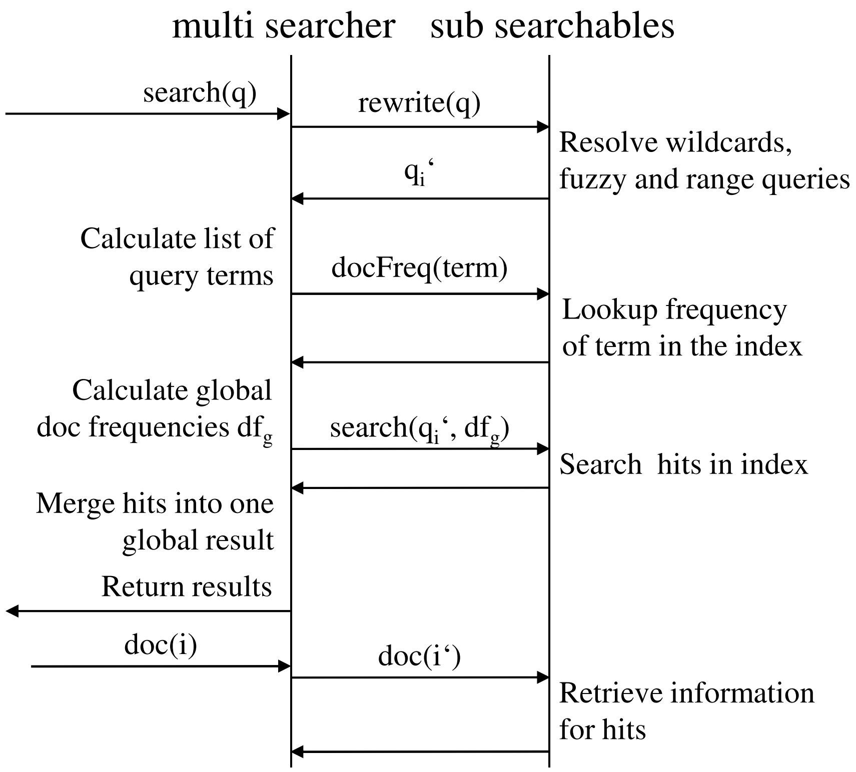

multi searcher
sub searchables
search(q)
rewrite(q)
Resolve wildcards, fuzzy and range queries
q_i‘
Calculate list of query terms
docFreq(term)
Lookup frequency of term in the index
Calculate global doc frequencies df_g
search(q_i‘, df_g)
Search hits in index
Merge hits into one global result
Return results
doc(i)
doc(i‘)
Retrieve information for hits

Figure 5.13: Distributed search

but simplifies adding documents as they do not have to be added to the one and only responsible shard.

- Other operations that modify the index are internal operations. E.g. deletion of documents is done automatically when they are over their retention period and when they are not on hold. These operations can be scheduled as necessary.

These properties open a wide range of possibilities to work with the shards and to organize them. It is for example possible to switch a set of shards to read-only and then to optimize them or merge them into a new and larger shard in the background. As soon as these processes are completed, their result can replace the original shards. It is also possible to move shards to a different search server to rebalance the load.

The search server is a stand-alone Java application that provides search access to a set of local full-text indexes using Apache Lucene. On startup it receives the list of local indexes as an input parameter. It then opens the local indexes and creates a ParallelMultiSearcher that allows querying all of them at once. Finally this multi-searcher is made remotely accessible by other programs using Java RMI.

To integrate the results of the different search servers we are also using the ParallelMultiSearcher. In this case it integrates the distributed search servers into one searcher.

On each search request the following steps are executed in order to process the query. The steps within Apache Lucene are also shown in figure 5.13.

- After receiving a search request, the web application first retrieves the set of available and relevant indexes from the database. With date-range restrictions or mailbox restrictions it may be possible to return only a subset of the indexes.
- The respective search servers are looked up in the naming registry of Java RMI. The result is a stub that can be used to remotely access the search server.
- A global ParallelMultiSearcher is created that subsumes the search servers gathered in the previous step.
- The query is sent to the ParallelMultiSearcher, which in turn sends it to all indexes for a rewrite.
- All the relevant terms for the query are determined and the document frequency for them is gathered.
- The global document frequency of the terms is computed.
- The query is sent to all indexes together with the global document frequency. The global document frequency is required in order to prune the query processing on the search servers.
- The local indexes generate their local hit list.
- The multi-searchers aggregate the results from the local indexes or the search servers and generate a hit list.
- The top hits are displayed to the user showing information retrieved from fields stored in the index.

This implementation of the distributed search has a laboratory state and is especially communication intensive. Up to now this implementation served our requirements. Yet resource monitoring during performance tests showed a high network and CPU utilization. Two major directions for improvements are:

1. A leaner communication mechanism than Java RMI to perform the remote requests. The generic serialization used by Java RMI generates larger data volumes on the network for each request. A simple query with four terms executed on only one search server already generated over 100 packets on the network.

Initial tests using Katta[13] which internally uses the inter-process communication mechanism from Hadoop showed that the network traffic can be reduced to 1/4.

2. A simplified ranking algorithm that does not require the collection of global statistics for each query. With a large document collection it should be possible to use a precomputed global document frequency statistic at each search server without reducing the quality of the ranking significantly.

In order to balance the load on the available nodes the assignment of the index to the nodes has to be changed. This requires either moving indexes from one node to another or storing the indexes on a shared file system. In the second case the indexes can be temporarily copied to the local disks in order to reduce the latency.

5.2.10 Performance measurements

In this section we present the results of performance measurements for the two most important catalog use cases: ingestion of new documents and search for archived documents. We thereby focus on the catalog only, thus we don't persist the e-mail in its original form. The content-based identifier is calculated, entries are made to the database, the plain text is extracted, and the plain text together with some metadata is added to the full-text index. The replication by the DHT was disabled in order to avoid that the disks in the given test system become a major bottleneck.

To measure the ingest performance the different data sets are archived using a varying number of nodes. The result of this measurement is shown in figure 5.14. It shows the aggregated throughput of an increasing number of nodes for the different data sets. For the "random domino" data set, which is presumably the most realistic one, the system scales very well with the number of nodes. With eight nodes about 90 e-mails are cataloged per second. With this data set the retrieval of the mails from the source as well as the processing of the large e-mails consumes most of the time. The overhead for the distributed catalog is negligible.

For the smaller e-mails in the "enron domino" set the scalability is still good. Yet a slight degradation is already visible with eight nodes. At this point 485 e-mails are cataloged per second. This leads to a considerable network traffic between the nodes, which in turn reduces the performance. Extrapolated to one day this corresponds to more than 40 million e-mails. For the "enron zip" data set, where the e-mails are stored in zip-files, the system reaches its limits. In these dimensions one should think about batching messages over the DHT as e.g. proposed in [Gho06].

[13] http://katta.sourceforge.net/

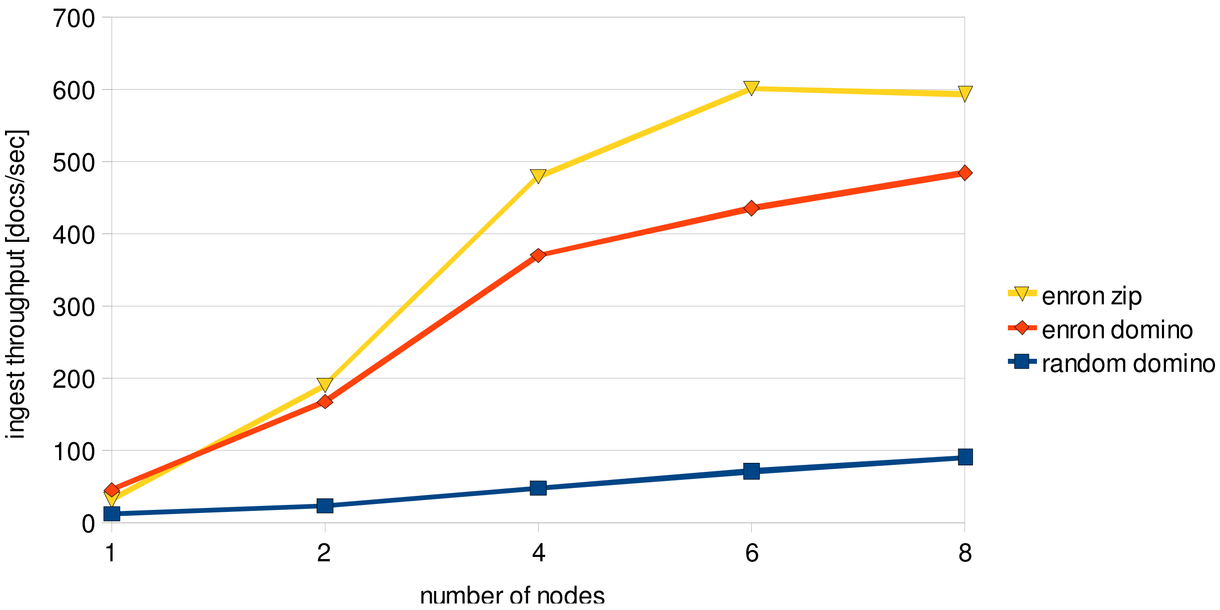


Figure 5.14: Scalability of the ingest process for the different data sets and an increasing number of nodes.

For the search performance measurements we used the large randomly generated data set stored on the domino server called “random large”. This data set was processed upfront by the respective number of nodes to create the database entries and the full-text index. The full-text index was optimized before the search tests. A user interface was started on each node and the load from JMeter was sent to them round-robin. Thus there was one node with JMeter, and a varying amount of nodes with the UI and the catalog running on each of them. Figure 5.15 shows that especially for higher numbers of users the maximum achievable throughput increases with the number of available nodes, and that the average response time decreases.

In the last test in this section we evaluated the effectiveness of pruning the number of search servers. In the concrete case we used partitioning by mailboxes. All e-mails from one mailbox are indexed within the index of one node. Thus with a mailbox restriction in the query, the query only has to be forwarded to this one node. The other queries still have to be processed by all four nodes in this case. The ratio of the user queries was mentioned when introducing the data sets in section 5.1.9.

The effectiveness of the metadata-based search server selection is shown in figure 5.16. All four measurements were done using four search servers. The blue lines represent the case where all four search servers where involved in each query (this is the yellow line in figure 5.15). The red line represents the case where only those search servers are contacted that have entries for the given user. As long as the nodes have

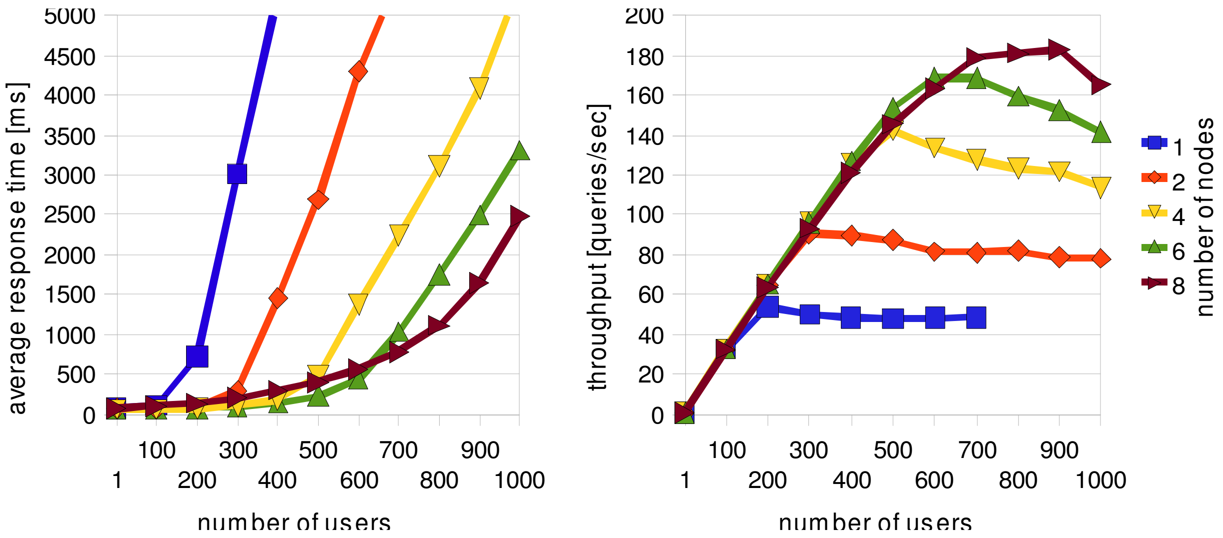


Figure 5.15: Scalability of the distributed search within the catalog

enough resources there is neither a positive nor a negative effect of the metadata-based server selection. But when the servers start to saturate, in this case at about 500 concurrent users, the benefits are clearly visible. The average response time stays below 0.4 seconds, whereas in the broadcast case it increases to more than 5 seconds. At the same time the system can serve about 100 additional requests per second.

The measurements have shown that the system provides a good scalability for the given range of nodes. Using a sharding-approach the scalability-problems of the individual systems can be avoided. The performance of the system can still be further increased by e.g.:

- Starting multiple ingestion processes on one node. Depending on the data sets the CPUs are only used to 50%. This is due to synchronization within the indexing process. With multiple ingestion processes and thus multiple indexes this can be avoided.

- Reducing the network utilization. The prime candidate here is replacing RMI in the search process with a leaner protocol like Hadoop IPC.

- Increasing the routing information available to the DHT. In such a small system compared to other P2P-systems it is possible to keep references to all the nodes and thus to avoid the $O(\log n)$ network hops per request.

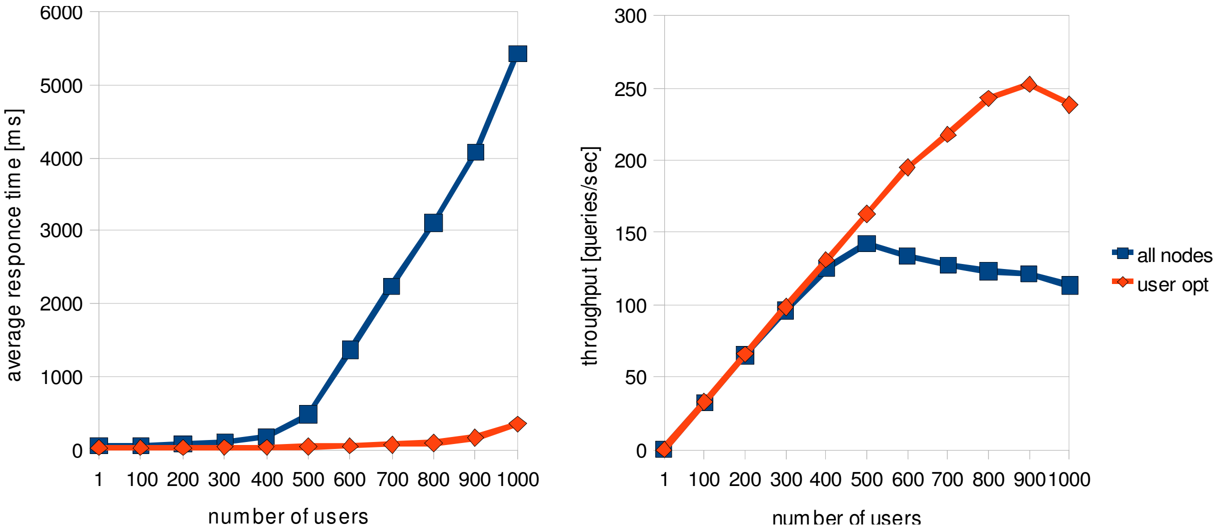


Figure 5.16: Effectiveness of metadata-based search server selection

5.3 Store and Retrieve

In this section we discuss storing and retrieving the original data. The ingestion of new e-mails always works in the same way: A data source, i.e. a user mailbox or a journaling mailbox is scanned and all qualifying e-mails are stored in the archive. They have to be stored in a way that allows reproducing them in their original form later.

The e-mails have to be stored persistently in the archive for a customer-specific amount of time. This retention period may range from a few months (e.g. 90 days) to several years (7 or even more) depending on the applicable laws and regulations. As one example rule 17a-4(b) from the US Securities Exchange Act of 1934 states, that "Every such broker and dealer shall preserve for a period of not less than 3 years, the first two years in an accessible place: [...] Originals of all communications received and copies of all communications sent [...] relating to his business as such.".

Strict regulations like SEC rule 17a-4(f) require that the records are archived in a "non-rewritable, non-erasable format". It does not require using write-once media like microfiche or CD-ROMs. Integrated hardware and software solutions using magnetic discs can be compliant, and in fact there are compliant systems on the market. Such a system prohibits specifically any password-protected way to delete records before their retention period is over. Detecting modifications by e.g. comparing fingerprints is not sufficient as the record in this case has already been modified.

This section starts with a discussion of different granularities of the stored artifacts. We then show how a flexible mechanism to store these artifacts using different back-end technologies was realized. The main part is about the different technologies and some implementation aspects. The section is concluded with performance measurements using the different technologies.

5.3.1 Dissecting the content

The documents have to be archived in a way that allows reproducing them later on, when they are needed. Yet this does not necessarily mean that each document has to be archived as one separate and self-contained entity. It can be an advantage to split the document into multiple parts and to store them as multiple entities. Of course this has to be done in a way that allows reassembling the document correctly when it is retrieved. Such a procedure can't be prohibited by a realistic regulation as at the end every file system stores a file as multiple separate blocks.

Before the e-mail reaches the step where it will be stored it was already processed in at least two ways: The identifiers for the e-mail and the different parts were computed,

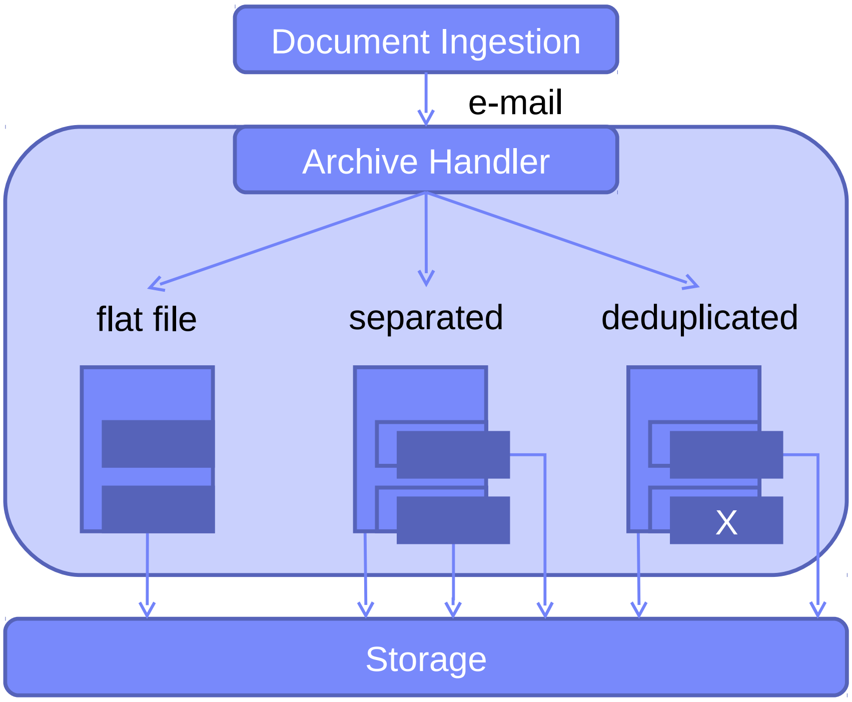


Figure 5.17: Alternative ways to store an e-mail in the archive

and a lookup was made for each part to check whether it is already in the archive. This information is available as metadata attached to the e-mail to be archived.

The pre-processed e-mail arrives at the ArchiveHandler shown in figure 5.17. This handler provides three options to store the e-mail in a storage system:

Flat file: With this option the e-mail is stored in its original form in the storage system. There is exactly one file per archived e-mail and this file is stored under the id of the e-mail. The advantage of this option is that an unmodified copy of the e-mail is stored in the archive which later allows an exact reconstruction. The downside is that it is not very space-efficient. For example the attachments in an e-mail with MIME are typically encoded with a Base64-encoding that only uses 6 bits per character and thus requires 33% more space than the binary form. As storage reduction is an important criteria of an e-mail archive, more sophisticated approaches are needed.

Separated: One downside of the flat file option is that it inherits all the drawbacks of the mail format and its extensions. Especially the restriction to allow only 7 bits per character, and among the remaining characters there are still some that should not be used. For an efficient storage utilization an inefficient encoding like Base64 should be avoided. To this end the different parts are extracted from the e-mail and are stored separately in the storage system. In this way they do not have to be encoded and all 8 bits can be used. The remaining stub consisting of the e-mail header and the headers of all the parts is stored as one additional entity.

Deduplicated: This extension of the previous option only stores a part if it does not already exist in the archive. The part marked with an "X" in the figure represents an attachment with an identifier that already exists in the archive. In a previous processing step this information was added to the e-mail. The part is not stored in this case, but instead a reference to the already stored part is used.

The advantage of the "flat file" approach is that fewer disk operations are necessary and that the data for one e-mail is grouped together. Yet it only allows a coarse grained access and thus more data may have to be retrieved than are actually required. The "separated" approach introduces a finer grained access to the data. But in exchange the number of disk operations increases. The main advantage of the "separated" approach is the reduction of the required storage capacity. Deduplication also reduces the number of disk operations during the ingestion. The number of disk operations should be between "separated" (no duplicates) and "flat file" (many duplicates) and depends on the number of duplicates in the data set.

Table 5.7 shows the results for the two storage options "flat file" and "deduplicated" for two different data sets. In all four tests the respective data set was stored in the local file system. With both data sets the amount of stored files increases with the deduplicated option. As a consequence of the increased number of disk operations and a slightly higher processing overhead the throughput decreases. The "enron zip" data set does not contain any duplicate e-mails. Duplication is therefore only done on the part level. This leads to a strong reduction of the stored data to about one third. Within the "random domino" data set every second e-mail has a duplicate. Therefore about one third of the e-mails have not been archived in both cases. The remaining reduction of the storage volume for deduplication on the part level is only about 5%. This shows that the effectiveness of the different options largely depends on the characteristics of the data set.

From a users perspective there are three different ways to retrieve an e-mail from the archive: 1) interactively viewing the e-mail in a web browser, 2) restoring the e-mail into the mailbox, and 3) exporting the e-mail into e.g. a PDF file for further review. In all three cases the different parts of the e-mail that are stored in the archive have to be retrieved and have to be reconstructed for the presentation.

When using the web interface of the EADM system the user in the first step uses the search interface to find the desired e-mails. As a response he receives a list of hits

data set	store option	throughput [docs/sec]	number of files	total size
enron	flat file	165	517431	24040200 (23 GB)
zip	deduplicated	141	776854	7452284 (7.2 GB)
random	flat file	12	66666	11603044 (12 GB)
domino	deduplicated	10	146013	10962596 (11 GB)

Table 5.7: Comparison of flat file storage and deduplicated storage for two data sets

containing several fields with metadata and a link to retrieve the e-mail. The retrieval of the e-mail starts when the user clicks on this link. On the next page the headers of the e-mail, one representation of the body, and a list of attachments with links to retrieve them are shown. In fact some of the attachments of the e-mail could be inlined images and could be directly displayed on this page, too. All other attachments are only retrieved in a further step when the user clicks on the respective link.

When the e-mail is stored with the "flat file" approach only one entity has to be retrieved from the archive. After that it has to be parsed and the required parts are extracted and displayed. Thus for every request the whole e-mail is retrieved, although only some parts of it might be sufficient. Even the inlined images will result in separate requests from the browser to the archive to retrieve these images. With the "separated" approach multiple parts are retrieved, but only the required parts have to be retrieved.

When restoring the e-mail into the mailbox or exporting it, the "flat file" approach is advantageous as in this case all parts of the e-mail are required. In this cast the retrieve of the whole e-mail does not contain unnecessary information.

After the description of how the e-mail is stored and retrieved, the major part of this sections discusses different back-ends to actually store the content.

5.3.2 Managing different storage types

In practice it can be desirable or necessary to use different storage back-ends at the same time. For example when not all the documents have to be archived similarly, but instead there are different requirements for different documents. For example when e-mails from mailboxes of the upper management must be stored in separate locations. During the ingest, properties of the mailbox or the e-mail can be interpreted and using a set of rules the storage back-end is selected. Or the back-end system might still be the same, but it is necessary to separate different sets of documents.

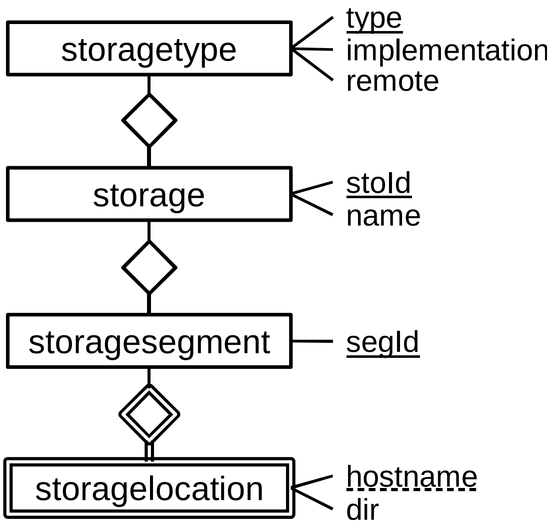


Figure 5.18: Data model to manage storage

In order to manage the different storage alternatives we implemented the data model shown in figure 5.18:

storagetype: A storage type represents a storage back-end. The "implementation"-property contains information that is used to load the driver for the storage type dynamically. Different implementations are discussed in the next section. The "remote"-flag is necessary to determine whether the storage can be accessed remotely. If it is false it can only be accessed from the node where the storage is located. If it is true the driver can also access it from a different node.

storage: Storage is a logical entity where the document is archived. It is referenced in the job to specify where the documents of this job should be archived.

storagesegment: One storage may subsume multiple storage segments. The identifier of the segment is recorded in the metadata of the e-mail e.g. in the index so that a retrieval from the right storage segment is possible later on.

storagelocation: A storage segment may be accessible at different locations. The storage location contains the necessary information the implementation needs to access a storage segment at a given node.

The interplay of these tables is illustrated using the examples shown in table 5.8. There are four different kinds storage in the example:

- stoId=1 represents local file systems (FS). There are two segments each located at one node (asbc05 and asbc06). On both nodes the storage is mounted as /data.

type	remote	stoId	segId	hostname	dir
FS	no	1	1	asbc05	/data
FS	no	1	2	asbc06	/data
CFS	no	2	3	asbc05	/shared
CFS	no	2	3	asbc06	/shared
RM	yes	3	4	asbc07	http://asbc07:9083/icm...
RM	yes	3	5	asbc08	http://asbc08:9083/icm...
Past	yes	4	6	asbc05	
Past	yes	4	6	asbc06	

Table 5.8: Example storage definitions

They are grouped together into one storage (stoId=1). Remote access to these file systems is not possible.

- stoId=2 represents a cluster file system (CFS). The cluster file system is mounted at two nodes. In this case both nodes have access to the same data. Therefore they both have the same segId 3.

- stoId=3 represents two separate resource managers (RM). Each resource manager manages its own segment (segId 4, 5) and has its own location. As the resource manager is based on a HTTP-server it can also be accessed from other nodes.

- stoId=4 represents a P2P system, in this case Past. All peers share the same segment and they are remotely accessible. The information is mainly used by clients to initially connect to the P2P system (bootstrap). It is also possible to define multiple DHTs in order to store disjoint collections of documents.

Each job to ingest e-mails from a mailbox into the EADM system includes amongst other parameters the identifier of the storage where to store the e-mails. Before a dispatcher submits a job to a worker, he first has to make sure, that the worker can access the specified storage. Either it must be remotely accessible or a segment must be available at the node where the worker is running. The identifier of the storage segment as well as the information about the location is sent together with the job description from the dispatcher to the worker. The worker then starts processing the job and stores all documents to the given storage location. The identifier of the storage segment will be stored together with the metadata so that the e-mail can be retrieved later on.

On retrieving an e-mail the system also has to look up a storage location where the given segment can be accessed. As this information is quite static it can be cached to reduce the lookup costs. In case the storage location is not local and the storage type is not remotely accessible we are using an HTTP-redirect to forward the browser to the right node.

5.3.3 Storage back-end systems

In this section we analyze different technologies to store the e-mails. The main categories we look at are:

- Local file systems (FS)
- Network file systems (NFS)
- Cluster file systems (CFS)
- Enterprise Content Management systems (ECM)
- Peer-to-peer systems (P2P)

The broad spectrum of technologies was chosen to get an overview of the different possibilities. Some of the systems differ remarkably regarding ingest, retrieve and management. In this section we first discuss the different systems at a higher level. In section 5.3.4 we then look at implementation details when using a subset of these systems as part of the EADM system.

5.3.3.1 Local file system

With this back-end the data is stored in a local file system that is exclusively mounted on one node. The file system can be any standard file system that is shipped with the operating system, like e.g. ext3 or ReiserFS on Linux or NTFS on Windows.

The data is stored in a file whose name is derived from the identifier of the data. In this way the data is stored in a unique location and it can be retrieved later on without requiring an additional mapping. However, file systems often have performance problems when there are too many files in one folder. To avoid this problem the files can be arranged in a folder structure. These folders can either be pre-defined, or they can be created dynamically depending on the number of files in each folder. A simple approach is to use a prefix of the identifier to determine the name of the folder where the data has to be stored.

When the file system resides on a local disk of a node, the availability of the data is not very high. Many kinds of hardware failures from hard disk drive over controller, motherboard, CPU, power unit to the network connection may render the data inaccessible. To reduce the risk of a complete data loss, the data can be striped over multiple local disks with an appropriate RAID-level. The fact that the file system can only be mounted by one node at a time does not exclude the possibility to mount it on another node in time shift. The SCSI-bus for example allows connecting two nodes to the bus so that in case of a failure at one node, the other node can take over the disks. This requires some coordination between the nodes, but increases availability, and thus it is a common approach for highly-available systems. The file system could also reside in a logical unit (LUN) of a storage area network (SAN). In this way many nodes can potentially mount the file system, but one has to make sure that only one node actually mounts it at any given time.

5.3.3.2 Network file system

A network file system makes the contents of a local file system available to multiple nodes by introducing an intermediary server. Multiple clients can mount the file system over the LAN at the same time. The best known network file systems are SMB/CIFS from Microsoft and NFS from Sun.

For storing and retrieving the data of an archive, a network file system can be used just like a local file system. As multiple processes on different nodes can access the file system concurrently, the access has to be coordinated. The server is responsible for the coordination of the requests from the different nodes and the provisioning of a consistent state. Yet with many clients the node serving the network file system may become a bottleneck.

5.3.3.3 Cluster file system

A cluster file system (CFS) also allows to mount the file system at multiple nodes at the same time, but the realization differs from a network file system. The most obvious difference is that the CFS can be mounted directly by multiple nodes at the same time without requiring an intermediary server. Synchronization of the directory structure and the files content between the nodes is done by the nodes participating in the CFS. The fact that the file system is part of a distributed system is made transparent to the application by the CFS. Examples of cluster file systems are GPFS from IBM[14], OCFS2 from Oracle[15], and GFS from Red Hat[16].

In the original form all the nodes must have access to the disk(s) containing the shared file system. They can be for example LUNs provided by a SAN system. In this case the network is only used to coordinate the access by the different nodes. Some

[14] http://www.ibm.com/systems/clusters/software/gpfs.html

[15] http://oss.oracle.com/projects/ocfs2/

[16] http://www.redhat.com/gfs/

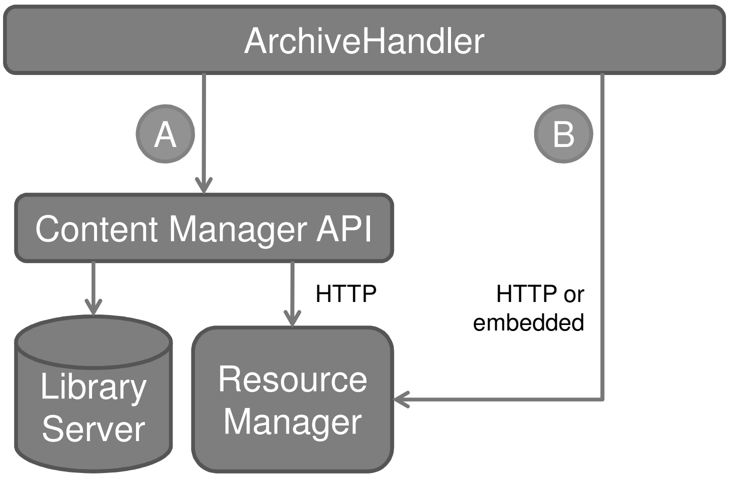


Figure 5.19: Two alternative approaches to integrate the IBM DB2 Content Manager

cluster file systems also support accessing the disks over the network when not all the nodes are directly attached to the SAN.

The distributed coordination makes the cluster file system more complicated than the network file system. But on the positive side the central bottleneck is avoided. Thus performance and availability are improved. Ideally, the failure of one node should have no effect on the other nodes using the cluster file system.

The EADM system can use a cluster file system just like a normal file system.

5.3.3.4 Enterprise content management systems

As already discussed in section 1.2, an ECM system provides many more functionalities than only storing and retrieving binary files. Yet we focus here on this functionality only. The concrete product we use is the IBM DB2 Content Manager V8.3 (CM).

Using CM is advantageous because it provides additional features that can be used. The two most important ones are replication of data between resource managers and hierarchical storage management to offload rarely used data to cheaper media.

The regular approach to use CM is to define a data model in the library server. For the EADM system this can be a very simple data model with only two properties for the data: an identifier and the binary content. With this approach the regular client API can be used to add, retrieve and delete the data. This approach is marked with "A" in figure 5.19.

The disadvantage is that CM performs its complete regular protocol for all operations. Additionally the metadata management capabilities of the library server are not used. Storing and retrieving binary data with a given identifier is the functionality that the resource manager provides on its own. Yet, all requests have to go through the library server in order to look up additional information, and for access control. It generates a token that grants the specific request to a resource manager.

Another approach to employ CM marked with "B" is to skip the API and the library server and to utilize the resource manager directly. The resource manager allows storing the content for a given identifier. At this point in the ingestion process no additional metadata management is necessary. Of course, this approach is not officially supported by the product, but a prototypical implementation is possible.

5.3.3.5 Peer-to-peer storage techniques

In the last years a broad spectrum of peer-to-peer (P2P) technologies was developed. A short overview can be found e.g. in [LCP$^+$05]. From systems to exchange music files like Napster, over systems to share computing resources like seti@home and folding@home, to systems with very specific features like censorship-resistant information sharing with freenet. We focus here on systems that allow to store files similarly to file systems.

Instead of having dedicated client and server roles, all participants of the P2P system work together in a peer-to-peer fashion. Each participant can be a client at one moment, and a server in the next moment. In this sense the system is symmetric.

A typical P2P system comprises thousands or millions of unmanaged peers. In such a setting new peers join the system all the time and others leave without prior notice. The system has to be able to deal with the high fluctuation (also called churn). In addition there can be faulty or even malicious peers. Information has to be replicated multiple times to prevent information loss. It is a challenging task to coordinate all the peers in such an environment and without a central instance.

One major distinction between different overlay networks is whether they are structured or unstructured. In a structured overlay network the location where a given piece of information belongs to is determined by the overlay network. Thus the overlay network determines the placement of information within the network. In an unstructured network there is no such placement strategy. Information can be placed anywhere in the system and the overlay network is responsible for the aggregation of the information from all peers to find the required information.

One application of structured overlay networks are the DHTs introduced in section 5.2.8. A hash key is used to identify the objects that are stored in the distributed system. When using a content-based hash value as the hash key they ensure the uniqueness of the identifiers and an even distribution of the identifiers within their name space. The identifiers defined in the EADM data model can either be directly used or mapped to identifiers of the overlay networks.

Some of the features that P2P systems provide are also interesting for ECM systems and archives. The most important ones being scalability, performance, security and self-maintenance. Therefore we also evaluated the applicability of P2P systems for the EADM system. Other features of different P2P systems like anonymity of the users or resistency to leechers are not further considered.

Although some systems may guarantee security, availability and privacy of the data even in a public system, the persons in charge of the archive very likely do not trust them. Not to mention the difficulties to convince an auditor that the system applies to the required regulations. Thus it is a more realistic approach to use dedicated machines and to set up a private and restricted P2P system on top. Such a system differs in several ways from "normal" P2P systems:

- The peers are in a managed environment where their availability is higher than the availability of anonymous peers in the Internet. Of course they may fail, too, due to defective hardware or software problems. But they are not switched on and off as often as personal computers.
- The number of peers in the system is much smaller. It has to be evaluated whether algorithms implemented by the P2P systems are still adequate in this different arrangement. A system that is build for millions of peers is not necessarily efficient with only tens or hundreds of nodes.
- Load balancing in P2P systems is often approached by randomly distributing the data over all peers. The more peers there are in the system, the smaller is the probability of a large skew. With fewer peers the probability of skew is higher and thus an active intervention might be necessary.
- The amount of data per peer is much higher, as most of the available space is dedicated to the EADM system. Thus moving the data to or from the peer requires a huge amount of bandwidth and thus takes some time.

For the evaluation we looked at the following five P2P systems in more detail. We will not describe the systems in detail, but instead focus on aspects relevant for the EADM system. A more detailed discussion of the systems is presented in the cited literature. We start the discussion of the P2P systems with systems that are designed to store and retrieve files. At the end we also have a look at a system that is focused on content distribution.

OceanStore

OceanStore [REG^{+}03, HKRZ02] was developed at the University of California, Berkeley. According to its web-page[17] "OceanStore is a global persistent data store designed to scale to billions of users. It provides a consistent, highly-available, and durable storage utility atop an infrastructure comprised of untrusted servers." OceanStore stores the files in a permanent and read-only form. Overwriting an existing file will result in a new version, and deleting files is not possible. OceanStore divides the peers into an inner ring and an outer ring. The nodes in the inner ring store the primary replica of the data and coordinate with each other to build a global and consistent state. This inner ring is implemented so that byzantine-faults are tolerated as long as at least 2/3 of the peers work correctly. The peers in the outer ring are used to store further replica of the data. An erasure code is used to increase the availability of the data.

Ivy

Ivy [MMGC02] is a P2P file system that provides its service as a local NFS server. Yet under the hood it works differently. The basic approach is to store log files in the distributed hash table (DHT) Chord. Each participant creates his own log file. On the one hand no locking is necessary when creating new files. On the other hand inconsistent updates can't always be avoided and have to be resolved with tools. In order to ensure authenticity of the log entries they are signed by the participant.

Multiple participants can aggregate their logs into views. When reading from the file system the logs are scanned to discover the required information. The participants can create snapshots of their view of the file system at some point in time to prune the scanning of the log files.

Files can be removed from an Ivy file system by appending a delete entry to the end of a log file. The log files themselves are never deleted.

Cooperative File System (CoFS)

Like Ivy the CoFS [DKK$^+$01] uses Chord to store the data. But instead of storing log entries in the DHT it stores blocks of the files. In contrast to storing complete files this has the advantage that files can be stored that would be otherwise too large for individual peers. In addition this results in a better load balancing. A new file is divided into several blocks and a SHA-1 hash is computed for each block. Each block is then stored in the DHT under its hash. In addition, a separate file containing the list of block hashes is generated to represent the file and it is also stored under its SHA-1 in the DHT. Similarly multiple hashes of such block lists can be stored in another file to represent a directory. The root node of such

[17] http://oceanstore.cs.berkeley.edu/info/overview.html, retrieved 01.04.09

a directory tree is stored in the DHT under the public key of the publisher and its content is signed by the publisher. In this way the authenticity of the data is guaranteed and only the original publisher is able to update data in the directory tree. Yet multiple directory trees can exist in one CoFS. The information in CoFS is stored for a specified time frame. The publisher has to ask for an extension when he needs the data longer. An explicit delete operation does not exist.

Past

Past [RD01a, RD01c, RD01b] is a large-scale, peer-to-peer archival storage facility that is built on top of Pastry. The files are identified by a combination of the SHA-1 of the files content, the public key of the publisher, and a random salt. The salt can be useful in two situations: 1) In the rare case of a collision of the SHA-1 of two different files. The identifier of one file can be changed by choosing a different salt. 2) In case an insertion fails because there is not enough storage capacity available at the responsible peers that hold the file and its replica. The insertion can be repeated on another set of peers by choosing a different salt. This movement of a file with all replica to a different region of the P2P system is one mechanism to tackle a major design goal of Past: Graceful degradation in case of a high utilization. The other mechanism is the redirection of an individual replica.

The system exports three operations: insert, lookup and reclaim. Insert stores the file with a customizable number of replica in the system and returns the above mentioned identifier. This identifier is required to retrieve the data, because it is not possible to look up a file by its SHA-1. The reclaim operation tries to free the space used by a file, but it does not guarantee to delete all replica.

Past stores the data and some metadata in an XML file in the specified directory. The downside of this approach is that not all characters can be used. Thus Past also uses the inefficient Base64 encoding to store binary data. The higher storage consumption is compensated by compressing the whole XML-file using gzip, yet at the price of needing even more CPU cycles.

With regard to security Past uses public keys and smartcards. For example the entries in the routing table are encrypted by the originating peer.

BitTorrent

BitTorrent is currently a very popular protocol for the distribution of large files over the Internet. Typical examples are ISO-images of CDROMs with operating systems, movies or television series. The BitTorrent client achieves a high throughput when downloading a file by loading different chunks of the file from different peers in parallel. Thus it is not bound to one serving peer and one TCP connection. The main contribution of BitTorrent is that after receiving some chunk of data the client immediately starts to serve this chunk to other peers and does not wait until the complete file is retrieved. In this way the load of serving a file is quickly shared between all the peers that already have some complete chunks of the file.

The set of peers that serve parts of a file, the seeders, is managed by a tracker. When uploading a file a user first has to set up this tracker. In the next step a torrent file is published for example on a web page describing the file and referencing the tracker. On downloading a file a client first retrieves the torrent file. In a second step the client connects to the tracker and retrieves the current set of seeders for this file. It then starts to download chunks from some or all the mentioned seeders. If some seeders are too slow or are not responsive at all, the client simply restarts downloading the affected chunk from another seeder.

In some sense there is not one BitTorrent system or network but there are some peers that are grouped together by one separate replica location service, the tracker, for each file. Although there are many trackers, one for each file, each tracker is a single point of failure. If a tracker fails, the managed file will be no longer available. The availability of the tracker node is not a feature of the BitTorrent protocol and thus has to be provided externally.

Another disadvantage of BitTorrent is that it only supports pulling files from the system. Adding new files mostly happens outside of BitTorrent. The tracker only appears when finally making the new file available. In addition there is also no active mechanism to replicate the data. A replica is only created when the file is downloaded to a peer on a user's request.

For the implementation we chose to use Past. Its provided features are closest to the required ones. It allows to persistently store and retrieve data, and it supports the expiration of the data. It manages the placement of the data within the system and does not use a complex directory structure. Additionally we implemented a system to store the content that is an extension of the Chord system.

5.3.4 Implementation details for some back-ends

In this section the handlers that were implemented to access some of the previously mentioned storage back-ends are described in more detail.

5.3.4.1 File system

The first handler stores the data into files in a file system. Any kind of file system that can be mounted at a node can be used. It could e.g. be a local file system like ext3, a cluster file system like OCFS2, or a network file system like NFS. The root directory, where the file system is mounted, is specified in the storage location table.

The name of the file where the data is stored is derived from the identifier of the data. In the concrete case the identifier is an MD5 hash computed over the complete e-mail or the part. The MD5 is a 128-bit number that is represented as a 32-digit hexadecimal value. The first two digits with 256 different values between 00 and FF are used as the name of a directory within the specified root directory. The remaining 30 digits are the name of the file within this directory where the data is stored. The following code excerpt in Java implements this behavior. The variable root is of type File and contains the root directory, and the variable id is of type String and contains the identifier of the document:

```
File dir = new File(root, id.substring(0,2));
if (! dir.exists()) dir.mkdir();

File file = new File(dir, id.substring(2));
os = new BufferedOutputStream(new FileOutputStream(file));
```

Using a fixed prefix of the identifier for the directory name reduces the problems of large directories with many files only by a constant factor. This factor can be increased by choosing a longer prefix, but at some point the increasing size of the root directory becomes a problem. A more sophisticated approach, not implemented here, is to recursively create sub-directories as soon as the size of a directory reaches a given limit.

Care should be taken that the values in the chosen prefix are evenly distributed. Especially they should not contain bits with a special meaning. This could lead to an imbalance in the size of the directories and thus to a performance degradation.

This implementation can be used with a cluster file system or a network file system without any changes. Yet, with multiple nodes accessing the same set of directories, the performance can degrade due to update synchronization within the file system.

Table 5.9 shows the throughput for the different file systems. In each test four nodes processed the data set concurrently, each of them having four threads. The only steps that were performed were the calculation of the content-based identifier and the actual store operation. As can be seen the local disks on the blades are clearly faster than the LUNs in the SAN. Even in the case where each node has its own LUN and file system.

file system	random domino	enron zip
local disk on each node	48.3	674.9
individual LUNs on one DS4100	34.0	86.0
OCFS2 with common root folder	29.2	53.8
OCFS2 with one root per node	27.9	63.2

Table 5.9: Ingest throughput with four nodes in documents per second for different file systems and data sets

The fact that all LUNs are defined on the same RAID array with seven disks limits the scalability. Especially with a high number of write operations. The cluster file system also defined in another LUN adds some additional overhead. Creating one root folder per node is only advantageous for the data set with the smaller e-mails where the rate at which files are created is higher.

5.3.4.2 Resource Manager

In a regular CM installation requests are sent via HTTP from the CM client API to the resource manager. A store request is sent as an HTTP post request with several parameters in the header of the request and the document in the body. A retrieve request is sent as a GET request where the document is returned in the body of the response.

Before the client API sends a request to store or retrieve a document to a resource manager, it first has to communicate with the library server. This is necessary in order to look up the location where the document is stored, and to authenticate the request. Only the library server knows the different users and what they are allowed to do. The library server returns a token to the client API that legitimates the request. Without a valid token the resource manager does not process a request.

In our first implementation we directly send the HTTP requests to the resource manager without using the client API or the library server. Of course we cannot send a valid token with the request. To circumvent this problem we use a flag that allows the resource manager to accept requests without a token. This is not viable in a production environment as it allows anybody to store, retrieve or delete documents from the resource manager. Yet, for a prototype it is a viable solution, and it should be possible to add the knowledge about the construction of the token for a product.

One problem is the mapping of the identifiers of the documents to the identifiers used by CM, the ItemIDs. These ItemIDs are defined as varchar(26) in the database. Our 32-digit MD5 hashes are too long for this column. The key to the solution of this problem is the very inefficient encoding of the MD5 hashes. They are encoded as hexadecimal numbers where each digit contains only 4 bits of information. After re-encoding the identifier with a Base64 encoding where each digit contains 6 bits of information it fits into the varchar(26) column. As the resource manager internally uses the ItemIDs as filenames, a variation of Base64 must be used where the encoded strings are valid file names. We also made sure that the sorting order of the identifiers with the two representations is the same. The mapping of the original identifier to an ItemId is a consistent and easily computable function.

The second implementation avoids the transfer of the data over HTTP and instead directly calls different methods within the resource manager. The prototype and the resource manager run in the same Java virtual machine. Of course this approach is not supported by the product and the used interfaces can and in practice do change between different versions. The internal changes between CM 8.3 and CM 8.4 are so immense that a complete re-implementation is necessary.

The expectation was that skipping the HTTP transfer reduces the average time to store a document in the resource manager. Yet it turned out that the improvement in comparison to the first implementation was only marginal. With four nodes, each running the prototype and a resource manager, a total throughput of 44 documents was achieved in both cases for the "random domino" data set. For the data set "enron zip" with the smaller e-mails the throughput increased from 374 documents per second to 447. Thus with larger data volumes the benefits of the shorter process are wasted by the data transfer.

Two clear advantages of the HTTP-based implementation are that it allows remote access and that the used interface is more stable. In fact our current implementation is no longer usable with recent versions of CM because the internal structure of the resource manager changed a lot.

A third option uses the catalog-operation of the resource manager. With that operation the resource manager does not store the file itself, but instead the file has to be stored within a folder by some external process before. It could e.g. be stored by the handler we use for file systems. The resource manager only adds an entry for the file in its database and after that manages it like the other approaches. The advantage is that only a small request has to be sent to the resource manager, the document itself is not transferred.

5.3.4.3 Past

For the implementation using the P2P system Past we use the variation with garbage collection (GCPast). The garbage collection regularly scans through all stored objects and removes objects that are beyond their life time. We use this garbage collection to implement the retention management of the archived e-mails.

We decided to cut back on the symmetry of the peers. The large amount of data in combination with the relatively small number of peers leads to a large re-balancing effort every time a peer joins or leaves. Therefore we decided to set up a P2P network with the storage peers only. The clients do not actually join the P2P network and thus the stored data does not have to be re-balanced.

The storage peers are explicitly started on the nodes. For the startup a peer needs a bootstrap address. This is the address of any node that is already part of the P2P network. This address is only an entry point to the network, and the address of any peer in the network is valid. Currently we maintain this list in the storage location table, but it could also be managed in the registry.

As the clients are no longer part of the P2P network, they need another way to insert and retrieve the data. To this end every storage peer in our system starts a Java-RMI server and registers it in the RMI-registry. A client connects to any of these RMI servers to store or retrieve data. Of course this adds an additional hop when storing or retrieving data. When ingesting new e-mails the storage peer is selected once at the beginning of each job (e.g. a mailbox) and all e-mails for this job use the same storage peer. When retrieving documents a storage peer is selected randomly for each operation. A timeout mechanism with retry on a different peer is not implemented yet. A more sophisticated approach is to regularly copy the routing information from a storage peer to a client similarly to Dynamo [DHJ$^+$07].

The following code excerpts show how a storage peer in Past is started. First the peer's host name is determined and some factories are set up.

```
// create a listening socket
InetAddress localAddress = InetAddress.getLocalHost();
NodeIdFactory nidFactory =
  new IPNodeIdFactory(localAddress, port, env);
SocketPastryNodeFactory f =
  new SocketPastryNodeFactory(nidFactory, port, env);
PastryIdFactory idf = new PastryIdFactory(env);
```

In the second step the underlying pastry node is created. The parameter bootaddress is the address of another peer that is already part of the P2P network. When the bootstrapping is successful the node joins the P2P network. If it is not successful, then the address of another peer can be tried.

```
// bootstrap the pastry node
NodeHandle bootHandle = f.getNodeHandle(bootaddress);
PastryNode node = f.newNode(bootHandle);
```

Now the persistent storage space is set up. As a parameter the root directory and the maximum capacity are required.

```
// initialize storage
Storage stor = new PersistentStorage(idf,
  "/local/storage/",              // directory name
  -1,                             // unlimited size
  node.getEnvironment());
```

The Past application is started using the previously generated objects. In the following example it has a 128 MB cache, creates one replica for each data object, and does the garbage collection of old objects once per hour.

```
// start the application
GCPast app = new GCPastImpl(
  node,
  new StorageManagerImpl(idf, stor,
    new LRUCache(
      new MemoryStorage(idf),
      128 * 1024 * 1024,           // 128 MB cache
      node.getEnvironment())),
  1,                                // one replica
  "",
  new PastPolicy.DefaultPastPolicy(),
  3600000l);        // garbage collect once per day
```

Past uses 160-bit SHA-1 hash keys to identify objects whereas our prototype currently uses 128-bit MD-5 hash keys. As a simple work around we calculate a SHA-1 hash over our MD-5 identifiers. For a production system a better integration is possible. The additional salt and public key ingredients of the generated identifiers as described earlier are not implemented.

5.3.4.4 Chord

The implementation based on Chord was already mentioned in section 5.2.8. It uses the partitioning and the routing mechanisms of OpenChord to forward our set of operations to the responsible node. The operations needed here, storing and retrieving content, are similar to the native operations of a DHT. With the extension of the operations it is principally possible to do e.g. text extraction or indexing at the responsible node.

The OpenChord implementation performs each operation in two steps: First the responsible node for the operation is determined, and then the operation is called at this node. There can be several nodes involved when determining the responsible node, but the call with the content as the payload is in most cases sent directly from the client to the responsible node. Only when the DHT has changed in the meantime and the formerly responsible node is no longer responsible the content has to be transferred again.

For the serialization of the content to be transferred the same mechanism is used like in the other cases to store the content. To actually store the content any back-end system can be used. After routing to the responsible node it can e.g. be stored in a local file system or in an embedded resource manager.

5.3.5 Comparison

In this section we go through several non-functional requirements and discuss how well the different systems support them. The security of the system and the data, respectively confidentiality, integrity and availability, are very important for an EADM system. Extensibility and performance are also examined.

5.3.5.1 Confidentiality

Confidentiality is the reason why a user is allowed to only see information from the system he is authorized to. It is obvious, that the information within the EADM system is at least partially confidential or personal or both. Thus it is very important that only authorized persons can read this information.

Albeit access control is mostly the duty of the catalog component of the system, it is still important, that the archived data can be protected from other users of the system.

A local file system allows to specify who can use a file or directory and in which way. Classical POSIX-compliant systems only allow specifying three different sets of permissions: one for the owner of the file, one for a configurable group, and one for everybody. Thus the possibilities are very limited with such a system. Newer systems also allow to specify access control lists with more than just three sets of permissions.

File systems typically distinguish three different permissions: read, write and execute. Within an EADM system nobody should have write permissions on the archived files, and execute permissions are not required at all.

A cluster file system tries to provide a normal file system on top of a distributed system. Thus the characteristics are similar to the local file system. Additionally consistent user identifiers and group identifiers on the participating nodes are necessary.

An ECM system provides a more detailed model for access control. The IBM DB2 Content Manager allows specifying the access control permissions either on an item type level, or on an item level. Thus one has to create a different item type for each set of items with the same user permissions, or one has to enable access control on the item level. The second option has a performance penalty, but especially for mailbox management the overhead to implement the first option is too big.

Our implementation skips the catalog of the ECM system and directly uses the resource manager. Access control on the resource manager is simple: every request with a valid token is processed. Before creating the token the catalog checked the user and his permissions. Therefore the resource manager does not have to know anything about users or permissions. At the moment the validation of the token is skipped because we can't generate it within the prototype, but technologically this should not be a problem.

P2P-systems provide a wide spectrum regarding confidentiality: from everybody can read everything to systems that heavily use cryptographic methods to secure the data. Even though the motivation is often to protect the owner of a node from legal consequences because of hosting some data, they can also be used to ensure the confidentiality of the data.

The Past system we used is one of the systems that can use a cryptographic system with public and private keys. In fact, it is their vision to use smartcards. This system is used internally to sign and validate all requests, but it can also be used to encrypt the data. There is a problem when using this in an EADM system: the one who ingests an e-mail, the crawler, is not the one who is allowed to retrieve it, the mailbox owner. And it is not only the owner who is allowed to retrieve an e-mail. The compliance officer e.g. must also be entitled to retrieve the e-mails. Thus a sophisticated management of the keys within the enterprise is necessary. Another issue are expiring keys, which make it necessary to re-encrypt all the data with the new key.

The implementation based on Chord forwards the request to the responsible node and therefore only has the content and its identifier as parameters. Thus user-specific actions are currently not possible. They would also depend on the chosen back-end.

5.3.5.2 Integrity

Integrity ensures that a user can only modify the system in a way he is authorized to. In an EADM system a user is not allowed to alter or delete any of the archived artifacts. There are only two processes that change the set of archived data: the crawler inserts new documents and the retention management removes expired documents.

As mentioned above, a local file system as well as the cluster file system have permissions that specify who is allowed to modify a given file. Similarly, the write permission on a directory specifies who is allowed to create a new file.

The content-based hash key that is stored in the catalog and that is also used as the basis for the file's name can be used to check whether the file still has the same content.

It is possible to increase the integrity by using specialized compliance storage devices like an IBM DR 550 or a EMC Centera Compliance Edition Plus. They use disks to store the data, but their software makes sure that the stored data is not changed before the retention period is over. They can be used like a network file system.

An ECM system also allows ensuring the integrity of the data. Extensions like a records manager can be used to further extend these capabilities. Also ECM systems can integrate the above mentioned compliance storage devices.

With P2P systems the integrity again depends on the usage of cryptographical signatures and encryption. The PAST system we use for example checks the signatures of the content before it accepts it. This prevents storing faked data in the system. PAST also does not overwrite already existing data.

For the Chord implementation the integrity completely depends on the chosen backend system.

5.3.5.3 Availability

Availability means the system can be used when it is needed. The discussion will be towards having the data available in a certain time, not necessarily immediately (high availability).

The availability of the local file system heavily depends on the availability of the used storage media. For example, if a simple hard disk is used, the failure of this disk will make the data unavailable. With the large amount of disks that are required to handle the storage volumes for an enterprise the failure of a disk one day is inevitable. Thus simply writing the data to disks is no option that will provide the necessary level of availability.

One common way to increase the availability of a disk-based storage system is to build a redundant array of inexpensive disks (RAID). There are different RAID levels defined. Most of them will increase the redundancy in a way so that the failure of one or even multiple disks can be tolerated. The price one has to pay is that more disks are needed for a specific amount of data and that more disk operations have to be executed.

A RAID can be implemented by different components. It can be implemented in the operating system, where no special hardware is required, but some share of the processor will be occupied. It can be implemented in the hardware on the controller.

Or it can be implemented in a separate device. Most SAN devices and NAS devices implement different RAID levels.

Even when the hard disk is still functional the data will become unavailable when another component of the system fails or doesn't work. E.g. when the operating system or some other required software is shut down, when the network adapter is broken, or when the power supply fails. Thus there should be a way to take over the disks from one node to the other. SCSI for example allows adding two hosts to one SCSI bus at any time as long as only one host actively uses a disk. Logical units from a SAN device can also be easily assigned to another node.

The availability of a cluster file system also depends on the disk system that is used. Some systems can provide redundancy of the data on their own. The file system GPFS from IBM allows to specify a replication level for the data. In that case the file system creates independent copies of a file on different disks. Thus it does not rely on or require a redundant disk system. The cluster file system we use here, OCFS2 from Oracle, does not provide such a capability.

When implementing an ECM system one typically uses a RAID system to store the online data. An additional tape library may be used for long-term storage. An ECM system can provide additional capabilities on top of them. The IBM DB2 Content Manager for example allows setting up replication between its resource managers. In this case the resource managers copy the data from one resource manager to the other on a scheduled basis. In this case recently added documents can be missing on the replica, but most documents can still be retrieved when one resource manager fails.

For a P2P system the failure of a node in the system is the normal case. Nodes leave the system all the time, often without prior notice. At the same time new nodes join the system. Thus mechanisms to increase availability are key for these systems.

To this end P2P systems create multiple copies of the data and distribute them on the available nodes. Due to the high fluctuation in the set of available nodes the system has to rearrange the copies constantly in order to maintain the required number of replica. The autonomous (re-)organization of the data is a feature that is rarely seen by the other systems discussed before.

5.3.5.4 Extensibility

The data volumes that an EADM system has to manage are very likely to grow constantly. In the first years, when only new documents are added and no expired documents are deleted, this is obvious. But as the average amount of e-mails as well as the average size of an e-mail both grow, the data volume within the EADM system will not reach a steady state. Therefore it is necessary that the system can easily be extended by additional storage capacity. And ideally this should be possible without a service interruption.

There are obvious limitations for the extension of file systems on local disks. If there is still unused space on the disk, the size of the partition containing the file system can be increased. Most file systems allow growing to a larger size when offline, and some like e.g. ext3 and ReiserFS on Linux can even grow when online. Yet this is limited by the size of the complete disk.

Nowadays most operating systems can be set up to use a logical volume manager (LVM). The LVM is an intermediary between the physical disks and the file systems. With an LVM it is possible to define logical devices that stripe over multiple partitions of different disks. Thus with an LVM it is also possible to increase the size of a file system beyond the size of a disk by adding additional disks.

Extending a cluster file system when the size of the underlying disk is increased is typically supported. With OCFS2 it is possible to use a device provided by a logical volume manager. With this it is possible to add more disks. But care must be taken that the definition of the logical devices stays consistent over all participating nodes. This functionality is not part of OCFS2.

In contrast GPFS has built-in support for multiple disks. The maximum supported number of disks ranges from a few thousands to several millions depending on the version of GPFS. Thus with respect to the size of the file system GPFS can scale to very large file systems.

The resource manager part of the IBM DB2 Content Manager has two components: a database and a system to store the binary data. Databases provide mechanisms to explicitly grow the size of their table spaces, or to dynamically take as much space as necessary. There are limits regarding the maximum size of a table or the maximum number of rows. Yet the entries in this database are rather small, so this is not a severe problem. To increase the storage volume for the binary data you can increase the size of the underlying file system as mentioned above. Another option is to add a new file system to the node and include it to the same logical group (workstation collection) in the resource manager. The resource manager even allows defining different groups. With an appropriate set of rules the resource manager can implement a hierarchical storage management where old content is for example offloaded to a tape library.

In a peer-to-peer system the resources can be increased by adding a new node, and resources can be removed by removing a node. P2P systems, due to their heritage in large and volatile environments, have to be able to deal with these changes. However there are challenges like supporting nodes with a wide range of resources or dealing with faulty and malicious nodes.

For the Chord-based implementation we look at the concrete implementation in

back-end	random domino	enron zip
local file system	48.3	674.9
cluster file system	29.2	53.8
CM (http)	47.7	377.4
CM (direct)	46.7	432.5
Past	32.9	124.3
Chord + RM	40.9	121.9

Table 5.10: Average throughput in documents per second when ingesting the data sets with the given back-end on four nodes

more detail in 5.4.3.1.

5.3.5.5 Performance

The performance of the different back-ends when processing new documents was measured by ingesting the two data sets "random domino" and "enron zip". Four ingest services on different nodes were used and each of them had four concurrent threads. In each case the e-mails were retrieved from the source, the content-based identifier was calculated and the e-mail was stored in the back-end. Table 5.10 shows the results of these measurements.

The two data sets were chosen to show the influence of the data set. The e-mails in the "random domino" have a representative size and are retrieved from a real e-mail server, whereas the e-mails in the "random zip" data set are small and can be retrieved very fast from their source, zip-files in the file system. With the "enron zip" data set, the fastest back-end is nearly 13 times faster than the slowest system, whereas with the "random domino" data set this factor is clearly below 2. This shows the problems some back-ends have with a high number of operations, whereas the data volumes are less distinctive.

The "local file system" is clearly the fastest back-end followed by "CM (direct)" and "CM (http)". In all three cases the ingest service stores the e-mails on the same node. In the first two cases the ingest of an e-mail happens within one Java virtual machine. In the third case the e-mails are sent over HTTP to a resource manager on the same node.

The slowest back-end is the cluster file system. To some extend this can be attributed to the shared SAN storage used in this case, whereas all other back-ends were using a local disk. Yet cluster file systems are often not specifically designed to handle the insertion of many small files.

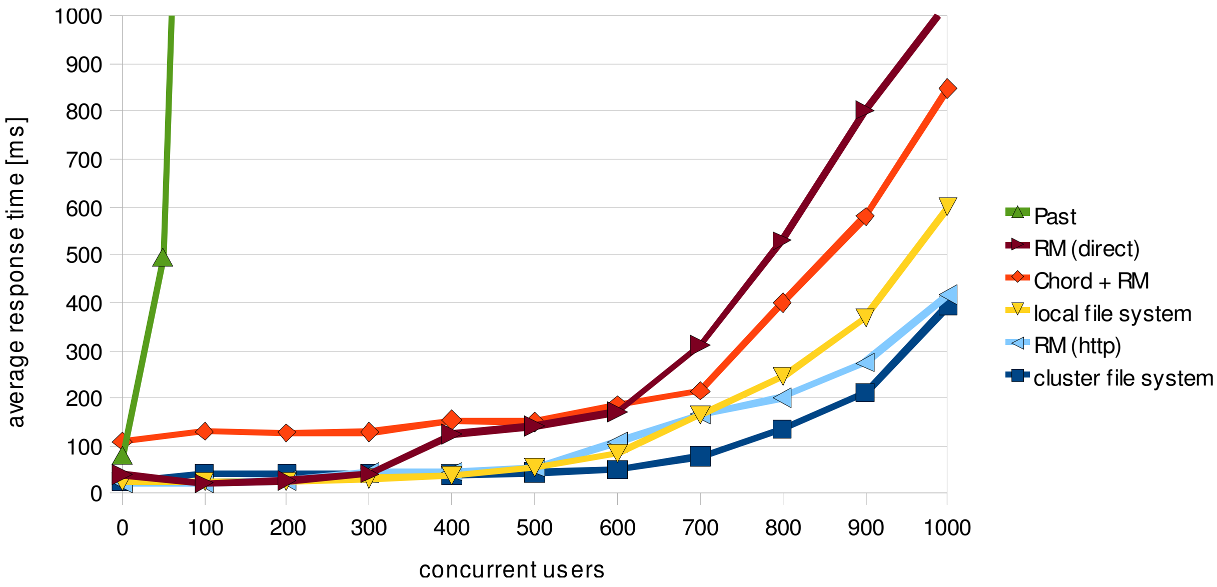


Figure 5.20: Retrieve throughput with four nodes and different back-ends

The two P2P systems are in the middle. With these systems an e-mail has to be transferred to a node different from the one where the ingest service runs on in three of four cases. This reduces the performance but it is necessary in order to maintain the partitioning.

For the retrieve performance tests we also created test plans for the Apache JMeter. In this case the simulated users retrieved either the e-mails or the attachments from the EADM system. Three different cases are defined:

1. The user retrieves an e-mail with text only. The identifiers of these e-mails were determined upfront and stored in a list. Only e-mails without an attachment were selected.

2. The user retrieves an e-mail that principally has attachments. But he only looks at the e-mail and doesn't retrieve the attachment.

3. The user retrieves an e-mail with attachments and after that also retrieves one of the attachments.

Case 1 is executed with a probability of 70%, case 2 with a probability of 20% and case 3 with a probability of 10%. Between the requests the users waits for 2.18 seconds in average.

Figure 5.20 shows the average times to retrieve e-mails and attachments from the different back-ends in a system with four nodes. We only show the results for the

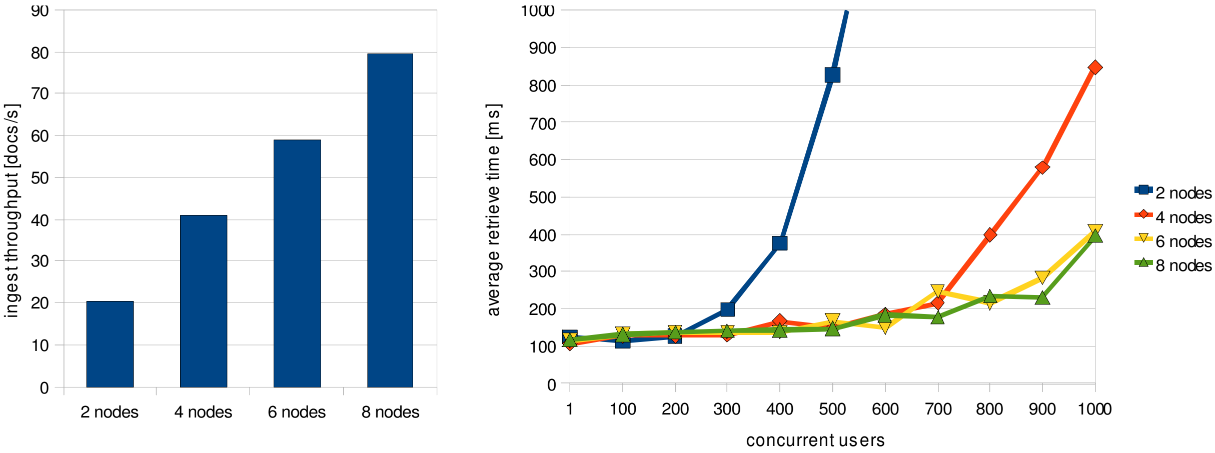


Figure 5.21: Scalability of ingest and retrieve with Chord + RM

"random domino" data set. With the small e-mails from the "enron zip" data set all back-ends were fast enough so that the node driving the tests with JMeter was the bottleneck.

In this case the Past system was clearly the slowest one. With only 100 concurrent users the average response time was already above three seconds. Tests with more users repeatedly failed due to a bug in the cache of Past. This problem did not occur with the small e-mails, so the longer period of time to retrieve the e-mail seems to be a problem here.

The two back-ends "RM (direct)" and "local file system" do not have the advantage of locality during the retrieve as during the ingest. To retrieve the content the client has to be redirected with a HTTP redirect to the right node. This obviously increases the response time. The "RM (http)" back-end, which only fetches the content from the right node, is faster for retrieves.

The winner for retrieves is the "cluster file system". This again shows the focus of this system on the read-only access for large files. It may also profit from caching and the separate fiber channel network.

The results for scalability measurements with an increasing number of nodes is shown in figure 5.21. In this test the content-based identifier was calculated for the content and it was archived over the Chord system into a resource manager with the direct implementation. The test was done with 2, 4, 6 and 8 nodes. On each node a Chord node ran as well as an ingest service and a user interface.

For the ingest the system scales well in the given range of nodes and the performance is comparable to the catalog operations measured in section 5.2.10. The retrieve tests show that despite some static noise the number of nodes doesn't have an impact on the average time needed for the retrieve operations. Thus distributing a large collection to multiple nodes should be no problem. Regarding the number of concurrent users the system roughly doubles the performance when going from two to four nodes. Yet with two more nodes the number of supported concurrent users does not increase that much. But this can be attributed to the test environment, as the CPUs on the client node running JMeter are completely utilized.

5.4 Scale-out and load balancing

In this section the data is supposed to be already distributed over several nodes. We now look at the dynamic arrangement of the data. The allocation of the data to the available nodes and possibly also the partitioning of the data is rearranged to better match the current situation. In the field, this rearrangement is inevitable for several reasons:

Growth: The amount of data managed by the system typically increases with time. To be able to handle the data, new resources at least in the form of storage capacity have to be added to the system. New nodes may also be necessary when the processing resources are no longer sufficient to handle the workload.

Decline: When a system is sized for the worst case like e.g. the Black Friday, it is vastly oversized most of the time. It is desirable to release the excess resources and to assign them to other tasks. In that case it is necessary to move the data on these nodes to nodes that remain in the system.

Load-balancing: No matter how thoroughly a system was planned, it is very likely, that over the years a skew of the utilization of the nodes evolves. Skew leads to a performance degradation and it should be avoided. By moving parts of the data from heavily loaded nodes to nodes with a lower load the load can be balanced.

Failure: Failures of parts of the system are unavoidable. In case of a failure the responsibilities of the node, and especially the managed data, have to be taken over by another node.

These reasons demonstrate why the system has to be flexible to integrate new resources and to remove existing ones. Yet, the system still has to act like one logical system. For the coordination of the data mapping to the responsible nodes, and the partitioning of the data, a repository virtualization layer (RVL) as shown in figure 5.22 is introduced. It implements a sharding approach, keeps track of a set of back-end repository nodes and maintains a global and consistent view of the repository. The main functions of this layer are:

- Routing the document-level CRUD operations to the right repository node(s). Depending on the defined partitioning criteria the document has to be stored on one specific node or on the other.

- Integrating new nodes into the system and removing nodes that are no longer required or available.

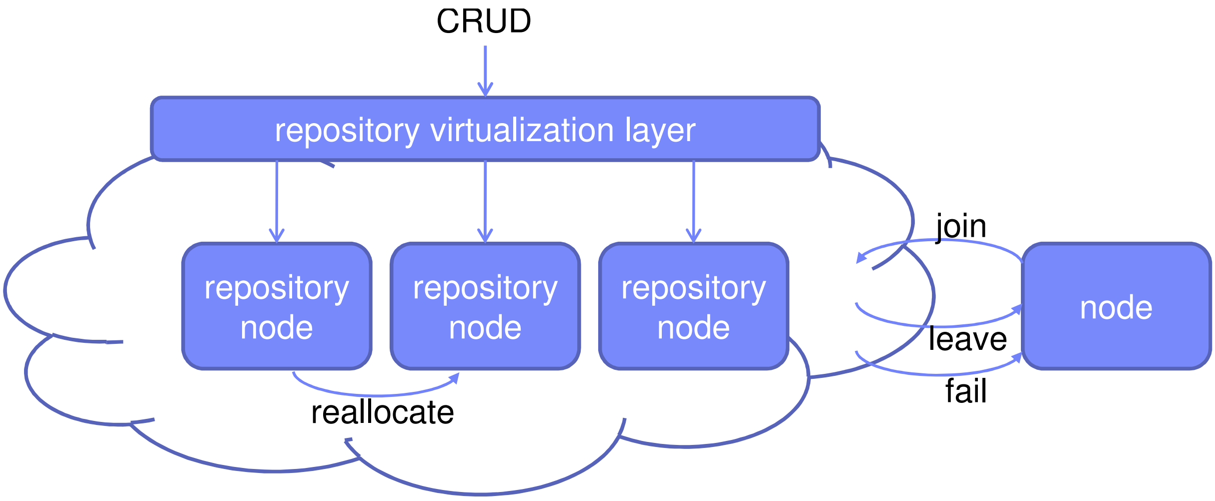


Figure 5.22: Repository Virtualization Layer

- Moving data between the nodes ensures a homogeneous system utilization. Especially, but not exclusively, when a new node has joined the system and has to take over its share of the workload.

The RVL is key to a dynamic and automated scale-out of the system. While new resources are added to the system it must still provide high throughput and low response times. The processes have to be automated with as little manual interaction as possible. This is the basis for a cost-efficient operation of the system: It is possible to start with a small test system and to add additional nodes on the go when necessary.

5.4.1 Use Cases for the Repository Virtualization Layer

In this section we discuss the primary use cases for the RVL that were implemented in [Moo08]. It is responsible for the stored files and the metadata stored in the catalog databases. The RVL is not yet responsible for the management of the indexes as they use a different partitioning.

5.4.1.1 Catalog database access

The RVL virtualizes multiple physical repositories into one logical repository. To this end it has to accept the CRUD-operations for the logical repository and has to forward them to the responsible physical repositories. One part of the repository is the catalog database. Thus all operations that access the catalog database have to be dealt with by the RVL. They are the following:

- insert document records
- insert member records
- insert part records
- expire documents
- create cases
- delete cases
- assign to cases
- remove from cases

The operations are implemented by a “remote service” that runs on each node. It provides local and remote access to the different catalog operations.

The partitioning of the data was chosen in section 5.2.6 in a way so that most of these operations, and especially the most frequently used operations, can be directly forwarded to one node only. Yet some operations have to access multiple nodes, e.g. a new case must be created on all the nodes as this table is fully replicated on all the nodes.

5.4.1.2 Store document

An important operation of the repository is to store new content. The input for this use case are the content and an identifier. The content is then stored persistently at some place. At which node the content is stored is defined by the partitioning scheme and is implemented as part of the RVL.

Transferring the content to the right node can be a huge effort. With an additional level of redirection this can be avoided. The content can be stored locally, and only a reference to the actual storage location is added to the responsible node. The downside is that now the distribution of the mapping as well as the distribution of the content has to be managed. Thus the system is getting more complicated. We therefore use this only as a temporary solution when e.g. large fractions of the content have to be moved.

5.4.1.3 Retrieve document

This use case is the inverse use case to the store document use case. With a given identifier the RVL retrieves the content that was stored under this identifier. Between the store operation and the retrieve operation the RVL may have moved or copied the content between different physical repositories. Yet this is transparent for the user of the RVL. The RVL always has to know how to get the actual data.

5.4.1.4 Add node

In a scale-out scenario more resources in form of processing power and storage capacity are added by adding new nodes to the system. The impact on the rest of the system, when integrating the new nodes into the RVL, should be as low as possible. Yet it might be necessary to rearrange the data which can have some impact on the already existing nodes. A slight performance degradation during the reorganization of the data can hardly be avoided due to the large amounts of data that have to be moved. But locking out the important ingest and retrieve operations for a longer period is not tolerable.

5.4.1.5 Remove node

After a peak, or when a large share of the content is removed from the system, there can be more resources in the system than necessary. In order to increase the cost-efficiency of the data center and to reduce unnecessary energy consumption it is desirable to remove nodes from the RVL in this case to use them by other systems or to shut them down. As the nodes host a lot of data that first has to be moved to other nodes, removing a node can be a time-consuming task.

5.4.1.6 Node failure

The system also has to be able to deal with the failure of nodes. In some cases it might be possible to remove the node before it actually fails but this is not always the case. With more nodes the likelihood that one of them fails is higher. Amazon realized during the operation of the Dynamo system, a system that is running in an environment similar to the one envisioned for large EADM installations, that most node failures are only of a temporary nature and the nodes come back online after some time [DHJ$^+$07]. In some cases of an overload or a lock the node may recover on its own, in other situations a reboot of the node may be necessary.

As discussed above, adding and removing nodes implies a large effort to rearrange the data. Starting this process every time a node fails will put an additional load on the system. Especially as most of the time the nodes recover with their data only slightly outdated, but still available, this seems unnecessary. To avoid this waste of resources we take an approach similar to Dynamo: A node is not automatically removed completely from the system when it is currently not available. It is only marked as not available so that requests are forwarded to another replica. But the data is not automatically reorganized. Instead, an administrator is informed about the problem. He may then decide based on additional information whether the node has a permanent problem or a problem that takes some time to fix. In some cases this could be automated by the monitoring and provisioning system. In either case the node is explicitly removed from the system and only then the system starts to rearrange the data. As the node has already failed in this case it is not possible to move the data away from the node before it leaves the system. To deal with such cases the RVL has to maintain replica of the data, and it has to recover these replica and rebalance the system. By not reacting immediately to a problem because of limited local information a lot of effort for rebalancing the system can be avoided. A failed node simply re-joins the system and is assigned to the same share of the data it had before.

5.4.2 Rearranging the data

The aim of our system is to support dynamic scale-out of a data-intensive content management application. If the data is statically assigned to a set of resources, this static assignment can become a constraint for scale-out. To avoid this constraint, the system must have the capability to move (parts of) the data from one node to another. Due to the large data volumes per node this movement may take a long time. It is therefore essential that it is done online without much interruption of other concurrent activities on the data.

As a starting point the data model stored in the relational database and described in 5.2.6 is used. The first implementation of this data model was only statically partitioned on two nodes (see [Rus08]). In [Moo08] the restriction of the number of nodes was removed and the movement of data between nodes was implemented. The particular challenge with this data is that it is managed in a DBMS that ensures ACID properties for concurrent transactions. Thus transactions could be locked which must be avoided or at least restricted. In this section we describe how the data is moved from one node to another.

The movement of the data takes place in three phases as depicted in Figure 5.23:

Phase 0: Before the move operation starts, the data is stored at the source node. This node is also responsible to perform all CRUD-operations for the data (creating new data, retrieving stored data, updating data and deleting data). The target node is not used yet.

Phase 1: The move operation starts with switching the responsibility for the data from the source node to the target node. In this synchronous copy phase essential

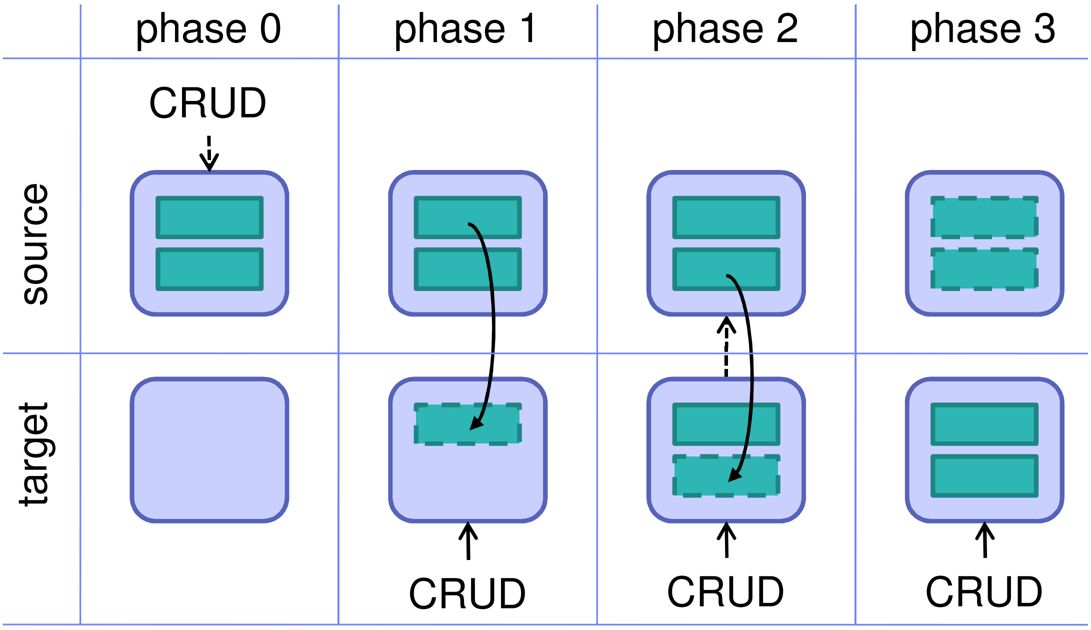


Figure 5.23: Moving data between repository nodes

data is copied to the target node. In the example this is only the small table with the cases. Although the target node is now responsible for the data, it does not have the required data yet to answer the CRUD-operations. They are therefore hold back and queued until phase 1 is over. Phase 1 ends when all the essential data is copied.

Phase 2: Phase 2 is the asynchronous copy phase. In this phase the data that was not yet copied in phase 1 is copied to the target node. But in contrast to phase 1 the system does no longer wait until this phase finishes. With the beginning of phase 2 the target node starts to process the CRUD-operations. In a first step the operations are executed locally. But as the copy operation is not completed yet, the necessary data might still be on the source node. In this case the operation is forwarded to the source node. It is possible that some data is inserted into the new target node that was not yet copied from the source node. Thus the data can't just be inserted into the new node, but instead it has to be merged.

Phase 3: In phase 3 the data is completely copied to the target node and it can process all the CRUD-operations on its own. The remaining task in phase 3 is to remove the copied data from the source node. This is done in small bunches as otherwise large delete operations lock the table for a long time and thus inhibit concurrent CRUD-operations.

The rationale for this relatively complicated move operation is to have a short interruption of the CRUD-operations. Only in phase 1 no CRUD-operations are performed at all. In all the other phases the system is processing them. Yet the activities of the move operations may have a negative impact on the performance of the CRUD-operations.

In the example all the data from the source node was moved to an initially empty destination node. But the mechanism also works when only a subset of the data is moved from the source to the target, and when there is already data on the target node. In practice this is more common than replacing a node completely. It is also possible to skip phase 3 to create a copy of the data. But in this case another mechanism to keep the copy in sync is necessary.

From the many different mechanisms to actually transfer data like exporting the data to files and importing them into other databases using utilities for backup and restore, replication, transferring tablespaces or federated access we chose federated access (see e.g. [Pur02]). It is a mechanism that makes a table stored in one database accessible within another database that is located on another node. This requires several steps: 1) the source database is registered to the database at the destination node, then 2) nicknames are defined in the destination database for all the required tables of the source node, and finally 3) these nicknames are used within SQL statements just like any local tables. At the end, when all the data is successfully moved from the source to the target node, the defined servers and nicknames are removed from the target database.

5.4.3 Realization of the use cases

In this section we describe how the different use cases for the dynamic arrangement of the system are realized.

5.4.3.1 Integrating a new node into the system

An automated integration of new nodes into the system is key to a scalable and cost-effective EADM system. In section 5.2.8 we showed how an existing Chord overlay network routes messages to a responsible node. In this section the steps that are necessary to integrate a new node into the system are discussed.

When a new node joins the overlay network it has to build up its own routing information, and the routing information of the other nodes in the ring has to be updated. First the new node needs the address of a so called bootstrap node, which is an arbitrary node being already part of the overlay network. The new node sends this bootstrap node a message with its own node identifier. The bootstrap node then does

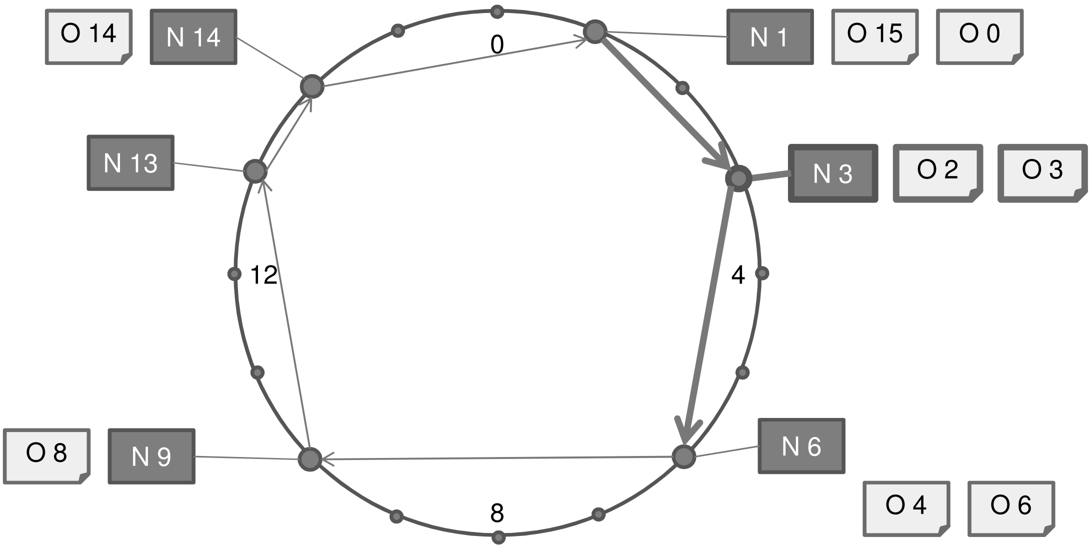


Figure 5.24: Example Chord ring after a new node N3 joined

a lookup for the identifier of the new node to find the place in the ring where it belongs to. For example a new node with identifier 3 belongs between the nodes N1 and N6 in figure 5.11. In the next step a successor of the new node the node N6 is set and the successor of node N1 points to the new node. The resulting chord ring is shown in figure 5.24. Stabilization tasks will adjust the finger tables of the nodes.

When a new node joins the DHT it has to take over a part of the data from other nodes. In the example the new node has to take over the objects O2 and O3. In a normal DHT environment the amount of data that has to be moved is not very large. The OpenChord-implementation therefore simply blocks the execution of any operations at the two affected nodes until all the data has been moved to the new node.

With the anticipated amount of data per node in an EADM system this movement of the data can easily take several minutes or even hours. Blocking even a part of the DHT for several minutes is not tolerable as in a system with high ingest activity very soon all the nodes are blocked.

Now the phased movement of the data in section 5.4.2 begins. After the new node joined the overlay network and thus is integrated into the routing of the messages it enters the synchronous phase 1. In this phase the new node already receives the requests for its range of the key space via the overlay network but it can't answer them yet as it does not have the required data. The requests are postponed to phase 2. In phase 1 the necessary subset of the data is moved to the new node. When the movement of the necessary data is completed, the nodes switch to phase 2. In this phase the larger part of the data is moved. The system already starts to process requests at the same time. As not all the data is yet available on the new node yet, the new node can't process all requests on its own. E.g. the lookup of an e-mail by the identifier may fail within the local database as its entry is still in the source database. To overcome this problem, all failed requests in phase 1 are forwarded to the source node. Only if the source database also has no entry for the identifier a negative answer will be returned. Otherwise the answer of the source database is returned. After all the data is moved and phase 3 is entered this forwarding of the requests to the source node is no longer necessary. This procedure enables the system to process requests most of the time even when new nodes join the system and the data has to be rearranged.

5.4.3.2 Removing a node from the system

When a node leaves the DHT another node has to take over responsibility for the data on the leaving node. In case of a Chord-DHT this is the successor node. Thus before the node actually leaves the system it has to move its data to its successor.

Similar to the joining case discussed above, the interruption of the request processing should be minimal. To this end the phased movement of the data is also used here. Before entering phase 1 of the move process the leaving node leaves the overlay network and thus makes its successor responsible for its share of the data. Yet during phase 1 the successor postpones the requests. Only the successor still has access to the leaving node to move the remaining data and to forward requests where the data was not moved yet. In phase 2 the successor starts processing requests for its new share of the data and if necessary forwards the requests to the leaving node. In phase 3 the leaving node is no longer needed by any active node in the overlay network. It can either be cleaned up or shut down immediately.

5.4.3.3 Node Failures

In practice nodes regularly fail without explicitly removing them from the system before. There are two important aspects to deal with these situations: 1) The stability of the overlay network, and 2) the availability of the data.

Regarding the availability of the data Chord does not store the data only on the responsible node. Instead the data is also stored on a customizable number k of successors. To increase the speed to generate the replica each node also maintains a list of its k successors. A background process takes care of maintaining the desired number k of replica in the system.

In addition to the necessary reference to the successor node, a Chord node also keeps redundant information about the overlay network in the finger table and the successor list. This information is mostly used to decrease the necessary routing steps but it is also used in case of failing nodes. If the successor of a node no longer responds to requests, the node can try to contact the successor of the successor. The necessary information is available in the successor list. The successor of the successor is responsible for all data that was stored on the successor when it failed. Thus the right node is contacted and the routing information can be adjusted. The replication strategy created a replica of the data on this node.

Moving the data from node to node in the EADM system takes a long time. In most cases a node fails only temporarily and re-joins the system when the data is still reorganized. To prevent unnecessary effort we use an approach similar to Amazon Dynamo [DHJ+07]. The overlay network will not trigger a rearrangement of the data when it detects a node failure. It starts to adjust its routing information so that all requests are sent to a replica of the node. Data is only moved on the explicit request of an administrator or by a higher level management system. This allows trying to reactivate a failed node.

5.4.4 Key Space Partitioning among Available Nodes

Load balancing is an important issue for a distributed system, especially when it only has a relatively small number of nodes that provide some service together. In an interactive system the slow performance of one overloaded system often has a bigger impact on the user's experience than the other fast systems. And for the EADM system the ingest services are throttled by the slow system. Therefore a relatively homogeneous utilization of the available nodes is desirable.

P2P systems do not have a central component and therefore there is no way to ensure a globally homogeneous load balancing. Some systems do provide mechanisms for load balancing within the surrounding of a node, but often they just rely on a good enough distribution over a large number of nodes. The fraction of the documents a node is responsible for is determined by the node's identifier and its neighbors' identifiers. In Chord, for example, a node is responsible for documents with identifiers between the node's identifier and the identifier of the predecessor node. The identifiers of the nodes and the documents are generated in a way so that they are scattered evenly over the entire key space, e.g. by using a good random number generator or a cryptographic hash function. On the one hand Godfrey e.a. [GS05] showed that the expected imbalance factor between the size of the segments on the nodes increases with $O(\log N)$ where N is the number of nodes in the system. But on the other hand the number of documents per segment decreases with $O(1/N)$. That means regarding the computational and network load the imbalance increases but regarding the volume of the stored data the imbalance decreases.

With a small number of nodes the average amount of data each node has to manage increases. And though the relative imbalance may be lower, the absolute difference can be larger. One outlier could randomly be assigned to a very large segment. In fact it could be too large for the node to store the data or to handle the induced workload. Thus in small systems one outlier can render the whole system overloaded.

One approach to tackle this problem is to artificially increase the number of nodes by introducing virtual nodes as recommended e.g. in [DHJ$^+$07, Gho06]. In this case every physical node hosts several virtual nodes. With $O(\log N)$ virtual nodes per physical node the imbalance between the segment sizes can be reduced to a constant factor [GS05].

An additional advantage of virtual nodes is that the number of virtual nodes a physical node hosts can be adjusted to the capacity of the node. Thus physical nodes with a higher capacity are assigned to more virtual nodes and vice versa. With the large annual growth in compute power and storage capacity, the nodes added to a system in one year will remarkably differ from the nodes added currently. Thus a production system has to deal with heterogeneous nodes.

When the virtual nodes of one physical node are replicated to different physical nodes, the additional load in case of the failure of a node is shared by multiple nodes. And as a further benefit a joining node can copy the data in parallel from multiple physical nodes.

Yet care must be taken that the virtual nodes do not thwart other properties of the system. When for example multiple replica of a document are stored on virtual nodes that are hosted on the same physical node the reliability statements are no longer valid. Also the length of the lookup path may increase with more virtual nodes.

In a first version of Amazon's Dynamo several virtual nodes with random identifiers were assigned to a joining node. With this technique they achieved an overall good load balancing across the available nodes even in the case of node failures. One problem of this strategy arose in production when a new node joined the network. Upon joining the network the new node has to acquire its share of the data from other nodes. The other nodes have to scan through their complete local data set and filter out the data that is now in the responsibility of the new node. This scan turned out to be an expensive task and resulted in the bootstrapping of a new node taking a long time. In the course of further development Amazon upfront divided the key space into Q equally sized partitions. Each one of the N physical nodes were then assigned to Q/N virtual nodes. In case of a node join or leave it is now possible to transfer the complete partitions and it is no longer necessary to scan through them. For a detailed description of these techniques refer to [DHJ$^+$07].

Due to the relatively small number of test machines in contrast to a regular P2P system we could not stick with the pure P2P system. The impact on the tests would have been too big. For the prototype we decided to start with a simple approach without virtual nodes and with a centralized assignment of the node identifiers. The identifiers are generated in a deterministic scheme where a new node is placed in the middle of one of the largest segments. An example is an identifier with three hexadecimal digits, the first generated identifiers are 000, 800, 400, C00, 200, 600, A00, E00, 100, ... With this scheme the largest imbalance occurs with three nodes, where one node is twice as large than the other two nodes. Of course, wholes from removed nodes should be reassigned to a new node before generating the next identifier in the sequence.

5.4.5 Data movement performance

In [Moo08] the performance of the rearrangement of the data within the database was measured. Initially the data from the "enron zip" data set was loaded into four nodes. In a first test a fifth node was added between two other nodes. As a consequence around 216,250 entries for documents, parts and members had to be moved from an existing node to the new node. The overall process took 336 seconds, from which 134 seconds were used to transfer the data, and 195 seconds were used to clean up the source database.

In a second test the new node was added while concurrently new e-mails were archived into the system. The data transfer time thereby increased to 182 seconds and the cleanup time increased to 327 seconds. This increase is only marginally due to the growing size of the database. A bigger influence has the necessary synchronization between the ingest and the transfer. Not only the transfer is slower, but also the ingest operations. During the transfer the ingest rate dropped from 30 documents per second to 7 documents per second, and during the deletion it dropped to 20 documents per second.

In the test only the catalog entries in the database were transferred to a new node. The index or even the archived files were not moved. It is obvious, that the movement of the data in a system with many files on few nodes is no viable option to deal with short term peaks. The movement of the data under the DHT should be seen as means to balance the utilization of the system on a wider time horizon.

For short term peaks [Moo08] also investigated adding only new ingest services that do not join the DHT. In this case the performance of four complete nodes and two nodes with ingest services only was comparable to six complete nodes. Thus for short term adaptations this seems to be a much better alternative.

5.5 Summary

In this chapter we looked at several implementation aspects of the proposed approach to implement an EADM system:

- In section 5.1 we first studied all the information an EADM system has to deal with and then developed a data model. For the implementation of this data model different technologies were analyzed. We finally chose to use a search engine to manage most of the metadata, and to use a relational database only for a small fraction that requires frequent updates. The search engine provides results with a good quality, and tests showed a good performance of this approach.

- In section 5.2 the focus was on distributing the data model onto multiple nodes. The aim was to support the scale-out of the system. By adding more resources it should be possible to support more users and/or a larger data set. First different methods to spread the data over multiple nodes were presented, before a partitioning scheme was proposed. To implement this data model we first looked at an implementation using distributes databases. But then we chose to implement the partitioning using a distributed hash table that assigns the documents to a set of otherwise independent systems consistently. With several measurements we showed that the scalability of this approach is very good.

- In section 5.3 the focus was on how to store the original data, that is the e-mails in our case. The first concern was the granularity of the objects to store. In a first case complete e-mails were stored, then in the more complex cases the parts were separated from the e-mails and were stored as separate files with the intent to reduce the overall storage consumption. We considered different technologies to store the content, from simple files over classical content management systems to P2P systems. Finally we measured and compared the performance of the different technologies. The scalability of the Chord-based implementation for the ingest was nearly linear, and the retrieve was also scalable and reached the capacity limits of the test driver.

- In the final section 5.4 we looked at the steps that are necessary to organize the data when either scaling out the system or when balancing the load. First we divided the data movement into several phases so that the time frame during a change when the system is not available is minimal. The implementation that was presented here organizes the metadata in the databases as it is determined by the distributed hash table. Performance measurements showed the times the system spends in the different phases, and uncovered some circumstances that have a significant influence on these times. Depending on the data volume the movement of the data can take a long time, yet the phased movement ensures that the time a node is completely locked is short.

6 Summary and Outlook

In this thesis we presented an approach to implement a scalable EADM system. The whole system was separated into different services that can be instantiated on multiple nodes. In this way the load generated by the application and its users can be spread among multiple nodes.

The services were implemented as web applications using the Java Enterprise Edition. Thus they can be deployed in different Java Application Servers. The services are stateful resources according to the Web Service Resource Framework. The standardized interfaces allow to use them by other components or third-party systems.

As EADM is a data-intensive application, the management of large amounts of data is key. Thus simply starting stateless services is not sufficient because the data used by the services has to be arranged accordingly. Exemplified by the index management and the information in the catalog database we showed how the data can be dynamically arranged.

6.1 An exemplary EADM implementation

To summarize this work and the different aspects that were discussed we look at one exemplary implementation of the EADM system. This implementation is based on lessons learned and discussed in the implementation chapter 5 of this thesis:

- The data is managed in a combination of a database and a search engine, where the larger share is inside the search engine. In the search engine we store whole documents because the de-duplicated approach has shown to be impractical for search operations (see section 5.1).

- The data is partitioned as described in section 5.2.6. This partitioning scheme allows to distribute the data over several nodes with a moderate internode communication. This is the basis for the horizontal scalablity of the system.
- To store the binary content we are using an extended resource manager. The extension allows us to store the database part of the data model in the database of the resource manager. By using the resource manager it is also possible to integrate hierarchical storage management.
- The repository virtualization layer implemented upon a distribute hash table allows to distribute the load on several nodes and to scale-out the system.

Figure 6.1 shows an example implementation of the system. In this example there are

- two ingest services that retrieve e-mails from the e-mail servers,
- three repository nodes that store the e-mails and metadata, two of them are collocated on nodes with an ingest service,
- one content service that makes the information within the archive accessible to the outside.

This implementation relies on a cluster file system between the nodes. It facilitates access to the files from all nodes. The cluster file system is used to store two different artifacts: The full-text indexes of the search engine and the original binary content.

The ingest service first locally creates a new full-text index which is then stored in the cluster file system where it can be loaded from by the content service. The services do not work directly with the files of the cluster file system but instead use copies on local disks because of the higher response times of the cluster file system. In both cases write access to the files containing the full-text index has to be coordinated by the system, so this is not an advantage when directly using files on the cluster file system.

The resource manager also stores the archived e-mails in files in a cluster file system. In this way all resource managers could in principle read all archived e-mails. Yet, each resource manager is only responsible for those e-mails that are assigned to it by the DHT, and where consequently the metadata is available in its database. A collision of filenames from different resource managers is not possible, as the filenames are based on unique identifiers, and they are uniquely assigned to the resource managers by the

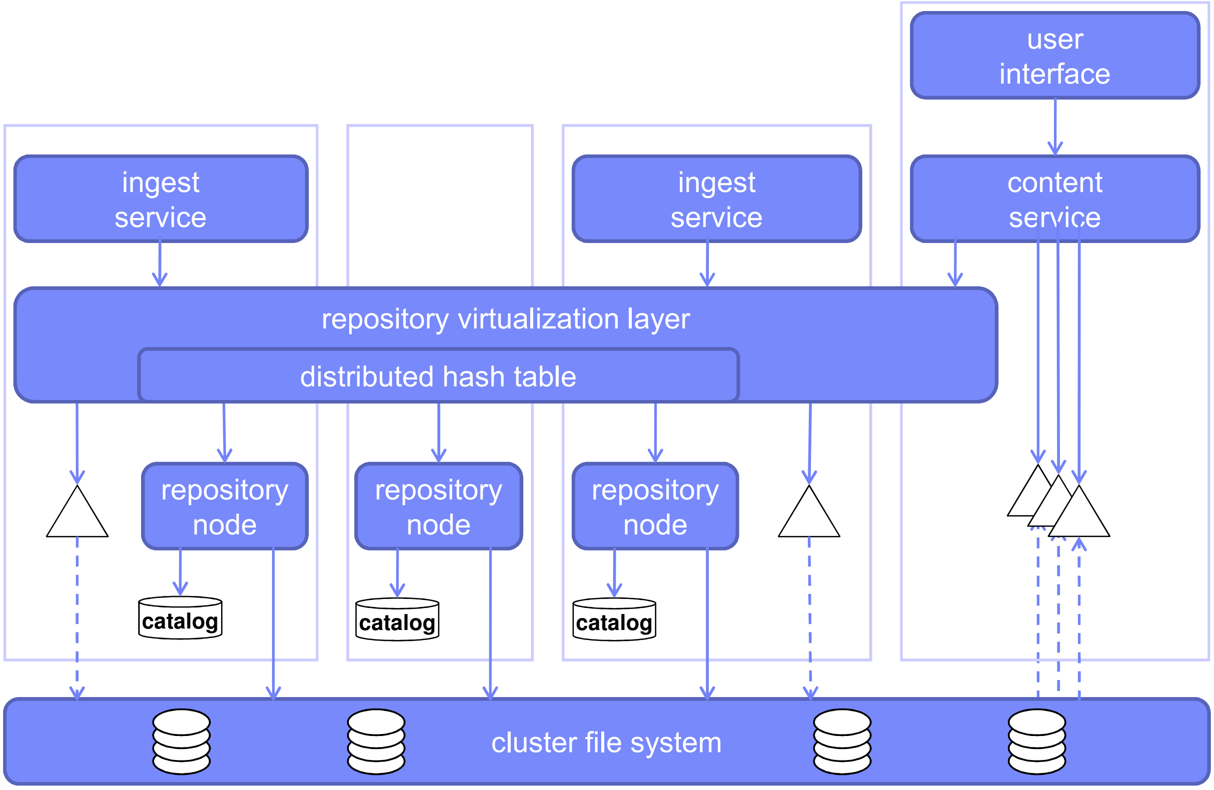


Figure 6.1: Sample implementation of an EADM system with four nodes

DHT. In this way it is possible to balance the load on the nodes by changing the DHT and relocating the metadata in the resource manager database. The archived e-mails themselves do not have to be moved as they are already accessible by all resource managers. Moving individual e-mails can also be critical when the regulations require an unalterable storage of the e-mails.

One downside of using a cluster file system is that the scalability of the EADM system could be limited by the scalability of the cluster file system. Especially the huge amount of archived e-mails and the number of disk operations to write and read them can become a problem. When the scalability of the cluster file system is not sufficient, another approach has to be chosen. One option are logical volumes in a storage area network that can be assigned to the repository node responsible for them at the moment.

The proposed approach for the implementation of an EADM system has several advantages:

- The separation of the system into different components allows to instantiate those components as required. Thus a relatively fine-grained assignment of components to the available resources is possible.

- The configurability of the components provides a flexible adaption of the system to different situations. For example the ingest service can perform a different set of steps for each job. This allows to enrich the searchable information in the index as necessary, and to steer the storage of the e-mails.

- The components are designed and implemented in a way that supports a scale-out approach. When some component is saturated another component of this type can be added to the system, and after a reconfiguration period this component takes over some load from the saturated component. On our test system the scalability of the ingestion of new documents, the search for documents, and the retrieval of the documents was shown.

6.2 Dynamic EADM on demand

The focus of this thesis are basic mechanisms to realize a scalable system. The different actions have to be triggered manually. Thus an administrator is required in order to adjust the system to a new situation.

In a productive environment, and especially in larger settings, the management of the system has to be supported by tools. Ideally the administrator specifies a set of service level agreements, and the system is configured automatically so that these agreements are fulfilled. Or less automatic, but sometimes preferred by customers, the system notifies the administrator of problems and offers different actions to fix them.

The dynamic management of the system is the topic of a related thesis. Figure 6.2 taken from [MKW^{+}09] shows how the EADM system is embedded into a framework that handles the dynamic provisioning of the system.

In the center of the figure the EADM system is depicted. The services studied in this thesis are shown in the upper part. But they represent only one part of an EADM system that has to be managed. The systems in the infrastructure software layer like databases, file systems, and indexes as well as the hardware in the bottom layer are also part of the EADM system and have to be managed too.

In order to be able to appropriately manage a system, information about the state of the system is required. This information is gathered with the real-time monitoring system depicted on the left side of the figure. To this end the implemented EADM services provide different metrics as Web Service Resource Properties. They provide general purpose metrics like CPU or disk utilization as well as application specific metrics like processed e-mails per second.

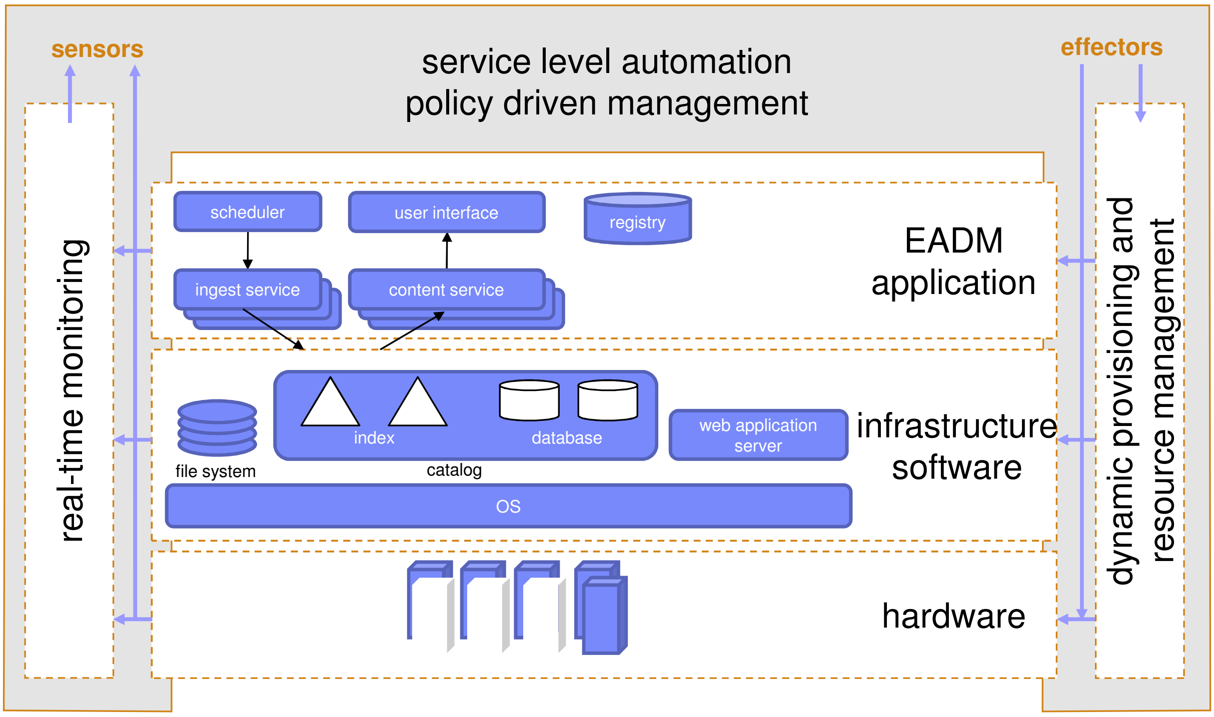


Figure 6.2: Dynamic provisioning of an EADM system (from [MKW+09])

The information gathered from the EADM system is analyzed by the service level automation and policy driven management. In a first step it detects existing or approaching violations of the guaranteed service levels. It also has to detect situations when the assigned resources are oversized and the current workload could be processed with fewer resources.

When an inappropriate configuration of the EADM system has been detected, the management system has to come up with plans on how to react to this situation. Therefore knowledge about the EADM system and the available infrastructure is required. This knowledge is stored in a resource model. An initial resource model for EADM was developed in [Ewe07].

The resource model describes the individual resources like nodes, operating systems, applications and services as well as the dependencies between them. For example to provision a new index service the following steps might be necessary: start a new node, install an appropriate operation system on the node, install the required software on the node, attach external storage, start the application server and finally deploy and start the service. The required order of these steps as well as all the parameters like IP-addresses, user names or the location of external storage are present in the resource model.

Finally the plans selected by the management system have to be executed in order to adjust the EADM system. This is where the effectors on the right hand side of figure 6.2 come into play. Typically these are shell scripts or small software components that perform the required actions on the nodes.

In this project the IBM Tivoli Service Automation Manager[1] is used as the basis for the management system.

6.3 Software as a service

Setting up the infrastructure and the software stack for an e-mail archive can be a huge effort. Once it is up and running it has to be managed in order to avoid a data loss and to deliver the service with the necessary quality of service. Additionally the system has to be adjusted to new or modified laws and regulations. Thus setting up and maintaining an e-mail archive can be an laborious task. Not to mention the fact that especially in small and medium business units the required skills may be missing.

This is the field for external service providers. They can build up the expertise for archiving and for compliance with the different regulations in different branches. And they can use these experiences for multiple customers.

According to the software as a service idea the customer orders the service from a service provider, adjusts parameters in the service level agreements as required, and pays the fees. The service provider sets up the infrastructure and software and has to provide the service with the guaranteed quality of service.

For a service provider it is important to use his infrastructure as efficiently as possible. But setting up a separate application with all required components for each customer can be expensive, too. Especially when the system for each customer has to be sized for the maximum guaranteed performance. Thus sharing resources between customers is an interesting option.

Of course, the EADM system operated by the service provider has to support multi-tenancy. It must be impossible that one customer can see information of another customer.

Different approaches are possible from using spare nodes for multiple customers, over instantiating services for different customers on one node to multi-tenant-aware databases. The discussion of all the options is beyond the scope of this work. Yet the service-based approach helps realizing a relatively fine-granular sharing of the resources.

[1] http://www-01.ibm.com/software/tivoli/products/service-auto-mgr/

6.4 Application to other content management applications

In this thesis we propose a service-oriented approach for the implementation of a scalable enterprise content management system. The concrete services we developed are aligned with one content management application: e-mail archiving, discovery and management. The result is not a complete framework which can be used directly in any content management application, because only a subset of the services within the ECM framework is used here.

The chosen application area has some specific characteristics that do not hold for every content management application. One of them is that the vast majority of the data within the system is never updated. The content is archived once and stays unchanged until after several years it is deleted. Only in the case when the archive moves to a new technology the content can be modified as part of a preservation activity. Another property is that an accurate consistency at any time is not important. E.g. content that was recently inserted into the archive does not have to be visible immediately.

We took advantage of these properties when designing and implementing the services. A good example for this is the construction and management of the full-text indexes. Both properties were exploited with the target to speed-up the ingest processing.

Other services like the distributed object storage using a distributed hash table to organize the partitioning are not dependent on these properties and thus can be applied in other content management applications, too. An enterprise content management framework is a way to share these common services between different application areas, and to use specialized services only where necessary.

Bibliography

[AFG$^+$09] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[Amd67] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the AFIPS conference*, volume 30, pages 483–485, 1967.

[Bau03] Michael Bauer. Design of an enterprise content management benchmark. Diploma thesis, University of Applied Sciences Würzburg-Schweinfurt, 2003.

[BBYRNZ01] Claudine Badue, Ricardo Baeza-Yates, Berthier Ribeiro-Neto, and Nivio Ziviani. Distributed query processing using partitioned inverted files. In *Proceedings of the 8th String Processing and Information Retrieval Symposium (SPIRE'01)*, pages 10–20, 2001.

[BFM$^+$03] Corinne Baragoin, Alain Fournil, Andrea Hirata Miqui, George Latimer, Susan Schuster, and Calisto Zuzarte. *Up and Running with DB2 UDB ESE: Partitioning for Performance in an e-Business Intelligence World*. Number SG24-6917-00 in RedBooks. IBM, 2003.

[Bis07] Malte Biss. Componentization and orchestration of content management services. Master's thesis, Universität Hamburg, 2007.

[BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. In *Computer Networks and ISDN Systems*, pages 107–117, 1998.

[Bre04] Eric Brewer. Combining systems and databases: A search engine retrospective. In M. Stonebraker and J. Hellerstein, editors, *Readings in Database Systems*. MIT Press, fourth edition, 2004.

[BYR99] Ricardo Baeza-Yates and Berthier de Araújo Neto Ribeiro. *Modern information retrieval*. Addison-Wesley Longman, 1999.

[CDY95] Surajit Chaudhuri, Umeshwar Dayal, and Tak W. Yan. Join queries with external text sources: Execution and optimization techniques. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California*, pages 410–422, 1995.

[Che03] Kenneth Chen. IBM DB2 content manager v8 implementation on DB2 universal database: A primer. Technical report, IBM, 2003.

[Cod70] E. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[Dad96] Peter Dadam. *Verteilte Datenbanken und Client-, Server-Systeme: Grundlagen, Konzepte und Realisierungsformen*. Springer, 1996.

[DC07] Carolyn DiCenzo and Kenneth Chin. Magic quadrant for e-mail active archiving, 2007. Technical report, Gartner, 2007.

[DHJ+07] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 205–220. ACM, 2007.

[DKK+01] Frank Dabek, M. Frans Kaashoek, David R. Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, pages 202–215, 2001.

[DRW07] Laticia Duboc, David S. Rosenblum, and Tony Wicks. A framework for characterization and analysis of software system scalability. In *Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 375–384, 2007.

[Ewe07] Christian Ewers. Basic functionalities of a grid-infrastructure for service-oriented content management. Master's thesis, Universität Hamburg, 2007.

[FB96a] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples. RFC 2049 (Draft Standard), November 1996.

[FB96b] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. RFC 2045 (Draft Standard), November 1996. Updated by RFCs 2184, 2231.

[FB96c] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. RFC 2046 (Draft Standard), November 1996. Updated by RFCs 2646, 3798.

[FB99] Armando Fox and Eric A. Brewer. Harvest, yield and scalable tolerant systems. In *Workshop on Hot Topics in Operating Systems*, pages 174–178, 1999.

[FK03] Ian Foster and Carl Kesselman, editors. *The Grid*. Morgan Kaufmann Publishers, 2nd edition, 2003.

[FKP96] N. Freed, J. Klensin, and J. Postel. Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures. RFC 2048 (Best Current Practice), November 1996. Obsoleted by RFCs 4288, 4289, updated by RFC 3023.

[FZRL08] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop (GCE'08)*, pages 1–10, 2008.

[Gho06] Ali Ghodsi. *Distributed k-ary System: Algorithms for Distributed Hash Tables*. PhD thesis, KTH Royal Institute of Technology, 2006.

[Gon09] Xiaowei Gong. Management of full-text indexes in a dynamic and distributed content management system. Master's thesis, Universität Stuttgart, 2009.

[GR93] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers, 1993.

[Gra81] Jim Gray. The transaction concept: Virtues and limitations. Technical Report TR 81.3, Tandem, 1981.

[GS05] Brighten Godfrey and Ion Stoica. Heterogeneity and load balance in distributed hash tables. In *24th Annual Joint Conference of the IEEE Computer* *and Communications Societies (INFOCOM 2005)*, pages 596–606. IEEE, 2005.

[GSGG02] Marlan Green, Sue Soy, Stan Gunn, and Patricia Galloway. Coming to TERM – designing the texas email repository model. *D-Lib Magazine*, 8(9), 2002.

[HD90] Hui-I Hsiao and David J. DeWitt. Chained declustering: A new availability strategy for multiprocessor database machines. In *Proceedings of the Sixth International Conference on Data Engineering*, pages 456–465. IEEE Computer Society, 1990.

[HG05] Erik Hatcher and Otis Gospodnetić. *Lucene in Action*. Manning, 2005.

[Hil90] Mark D. Hill. What is scalability? *ACM SIGARCH Computer Architecture News*, 18(4):18–21, 1990.

[HKRZ02] Kirsten Hildrum, John D. Kubiatowicz, Satish Rao, and Ben Y. Zhao. Distributed object location in a dynamic network. In *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, pages 41–52, 2002.

[JW00] Prasad Jogalekar and Murray Woodside. Evaluating the scalability of distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 11(6):589–603, 2000.

[Kam06] Ulrich Kampffmeyer. ECM – enterprise content management. white paper, Project Consult Unternehmensberatung, 2006.

[LCHC$^+$07] Jenny Li, Lisa Case-Hook, Ira Chavis, Tom Conway, Leonard Fox, Yvette Pegues, and Susan Williams. *IBM Enterprise Content Management and DR550 for e-mail Archiving and Records Management: Overview*. IBM Redbooks, 1st edition, 2007.

[LCP$^+$05] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys and Tutorials*, 7(2):72–93, 2005.

[LZW04] Nicholas Lester, Justin Zobel, and Hugh E. Williams. In-place versus re-build versus re-merge: Index maintenance strategies for text retrieval systems. In Vladimir Estivill-Castro, editor, *Twenty-Seveth Australasian Computer Science Conference (ACSC2004)*, volume 26 of *Conferences in*

Research and Practice in Information Technology, pages 15–22. Australian Computer Society, 2004.

[MA06] Luiz Rodolpho Monnerat and Cláudio L. Amorim. D1HT: a distributed one hop hash table. In *IPDPS*. IEEE, 2006.

[ME04] Bethany McLean and Peter Elkind. *The smartest guys in the room: the amazing rise and scandalous fall of Enron*. Penguin Group, 2004.

[MG05] Peter Merz and Katja Gorunova. Efficient broadcast in P2P Grids. In *Proceesings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05)*, pages 237–242, 2005.

[MHW06] Soumyadeb Mitra, Windsor W. Hsu, and Marianne Winslett. Trustworthy keyword search for regulatory-compliant records retention. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea*, pages 1001–1012. ACM, 2006.

[Mic07] Sun Microsystems. Messaging compliance and content management: A technical view. Technical report, Sun Microsystems, 2007.

[MKW⁺09] Cataldo Mega, Kathleen Krebs, Frank Wagner, Norbert Ritter, and Bernhard Mitschang. Content-management-systeme der nächsten generation. In Frank Keuper and Fritz Neumann, editors, *Wissens- und Informationsmanagement: Strategien, Organisation und Prozesse*, pages 539–567. Gabler, 2009.

[MM02] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric. In *Electronic Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, 2002.

[MMDS⁺07] José E. Moreira, Maged M. Michael, Dilma Da Silva, Doron Shiloach, Parijat Dube, and Li Zhang. Scalability of the nutch search engine. In *Proceedings of the 21st annual international conference on Supercomputing*, pages 3–12, 2007.

[MMGC02] Athicha Muthitacharoen, Robert Morris, Thomer M. Gil, and Benjie Chen. Ivy: A read/write peer-to-peer file system. In *OSDI*, 2002.

[MMSW07] Maged Michael, José E. Moreira, Doron Shiloach, and Robert W. Wisniewski. Scale-up x scale-out: A case study using nutch/lucene. In *Parallel and Distributed Processing Symposium*, pages 1–8, 2007.

[Moo96] K. Moore. MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text . RFC 2047 (Draft Standard), November 1996. Updated by RFCs 2184, 2231.

[Moo08] Alexander Moosbrugger. Evaluation of scale-out characteristics of an ecm system using overlay-techniques to support dynamic topologies. Diplomarbeit, Universität Stuttgart, 2008.

[MWM05] Cataldo Mega, Frank Wagner, and Bernhard Mitschang. From content management to enterprise content management. In *Datenbanksysteme in Business, Technologie und Web, 11. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), Karlsruhe, 2.-4. März 2005*, pages 596–613, 2005.

[MWZBY07] Alistair Moffat, William Webber, Justin Zobel, and Ricardo A. Baeza-Yates. A pipelined architecture for distributed text query evaluation. *Information Retrieval*, 10(3):205–231, 2007.

[Neu94] B. Clifford Neuman. Scale in distributed systems. In Thomas L. Casavant and Mukesh Singhal, editors, *Readings in Distributed Systems*, pages 463–489. IEEE Computer Society Press, 1994.

[Pfi98] Gregory F. Pfister. *In Search of Clusters: The Battle in Lowly Parallel Computing*. Prentice Hall International, 2nd edition, 1998.

[Plo04] Jeffrey Plotkin. E-mail discovery in civil litigation: Worst case scenario vs. best practices. white paper, kvsinc.com, apr 2004.

[Pur02] Micks Purnell. Fundamentals of IBM DB2 federated server an relational connect. Technical report, IBM Data Management Advances Technical Support, jun 2002.

[Rad05] Radicati Group. Hewlett-Packard: Taming the growth of email - an roi analysis. Technical report, Radicati Group, Inc., 2005.

[RD01a] Antony I. T. Rowstron and Peter Druschel. Past: A large-scale, persistent peer-to-peer storage utility. In *HotOS*, pages 75–80. IEEE Computer Society, 2001.

[RD01b] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In Rachid Guerraoui, editor, *Middleware*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350. Springer, 2001.

[RD01c] Antony I. T. Rowstron and Peter Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, pages 188–201, 2001.

[REG$^+$03] Sean Rhea, Patrick Eaton, Dennis Geels, Hakim Weatherspoon, Ben Zhao, and John Kubiatorwicz. Pond: the oceanstore prototype. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03)*, mar 2003.

[Res01] P. Resnick. Internet Message Format. RFC 2822 (Proposed Standard), April 2001.

[RS04] Venugopalan Ramasubramanian and Emin Gün Sirer. Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays. In *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI 2004)*, pages 99–112, 2004.

[Rus08] Martin Russold. Erweiterung des Datenmodells für ein System zur Email Archivierung und Discovery. Diplomarbeit, Universität Stuttgart, 2008.

[RWMS04] Arcot Rajsekar, Michael Wan, Reagan W. Moore, and Wayne Schroeder. Data grid federation. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA '04*, pages 541–546, 2004.

[Sch06] Sergej Schütz. Indexierung von E-Mail-Archiven mit hohem Nachrichtenaufkommen. Diplomarbeit, Universität Stuttgart, 2006.

[SE05] Brian E. Seward Esq. Email discovery: Tape is not enough. *AIIM E-DOC Magazine*, sep/oct 2005.

[SMK$^+$01] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2001.

[Spu07] Bob Spurzem. Mimosa nearpoint for microsoft exchange server. white paper, Mimosa Systems, Inc., 2007.

[SQL03] Information technology - database languages - SQL, ISO/IEC 9075-*, 2003.

[Sto86] Michael Stonebraker. The case for shared nothing. *IEEE Database Engineering Bulletin*, 9(1):4–9, 1986.

[Sun00] Sun Microsystems. Javamail API design specification, version 1.2, 2000.

[Sym06] Symantec. Symantec enterprise vault introduction and planning. Technical report, Symantec, 2006.

[Thi04] Graeme Thickins. Compliance: Do no evil – critical implications and opportunities for storage. *Byte and Switch Insider*, 2(5), 2004.

[U.S06] U.S. Department of the Interior. It's in the mail: Common questions about electronic mail and official records, 2006.

[Vog08] Werner Vogels. Eventually consistent. *ACM Queue*, 6(6):14–19, October 2008.

[WCL+05] Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, and Donald F. Ferguson. *Web Services Platform Architecture*. Prentice Hall, 2005.

[WK83] Eugene Wong and Randy H. Katz. Distributing a database for parallelism. In David J. DeWitt and Georges Gardarin, editors, *SIGMOD Conference*, pages 23–29. ACM Press, 1983.

[WKM+08a] Frank Wagner, Kathleen Krebs, Cataldo Mega, Bernhard Mitschang, and Norbert Ritter. Email archiving and discovery as a service. In Costin Badica, Giuseppe Mangioni, Vincenza Carchiolo, and Dumitru Dan Burdescu, editors, *Intelligent Distributed Computing, Systems and Applications, Proceedings of the 2nd International Symposium on Intelligent Distributed Computing - IDC 2008, Catania, Italy, 2008*, volume 162 of *Studies in Computational Intelligence*, pages 197–206. Springer, 2008.

[WKM+08b] Frank Wagner, Kathleen Krebs, Cataldo Mega, Bernhard Mitschang, and Norbert Ritter. Towards the design of a scalable email archiving and discovery solution. In Paolo Atzeni, Albertas Caplinskas, and Hannu Jaakkola, editors, *Proceedings of the 12th East European Conference on Advances in Databases and Information Systems (ADBIS)*, number 5207 in Lecture Notes in Computer Science, pages 305–320, 2008.

[WY05] Akitsugu Watanabe and Haruo Yokota. Adaptive overlapped declustering: A highly available data-placement method balancing access load and space utilization. In *Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, 5-8 April 2005, Tokyo, Japan*, pages 828–839, 2005.

[Yun05] Wayne W. Yung. Explore the IBM mail management and compliance solution. *developerWorks*, 2005.

[ZBB+06] Wei-Dong Zhu, Satya Bhamidipati, Elena Bogdanova, Toralf Gerstaecker, Ryan Lott, and Donald L. Wilcox Sr. *Best Practices for Setting Up an IBM CommonStore Solution For Mailbox Management, e-Mail Retention and Discovery*. Number SG24-7325-00 in RedBooks. IBM, 2006.

[ZFH+06] Wei-Dong Zhu, Torsten Friedrich, R Hogg, Juergen Maletz, Philip McBride, and Dean New. *E-mail Archiving and Records Management Integration Solution Guide*. Number SG24-6795 in RedBooks. IBM, jan 2006.

[ZGK+06] Werner Zurcher, John Glen, Etsuko Kagawa, Julie Murakami, Rich Alford, Christina Baribault, and Jeannette Starr. Introducing symantec enterprise messaging management for microsoft exchange. Technical report, Symantec, 2006.