Scalability Evaluation of a P2P Content Distribution System

Serge Abiteboul
INRIA-Orsay
fname.lname@inria.fr

Radu Pop
INRIA & Mandriva
fname.lname@inria.fr

Gabriel Vasile
INRIA-Orsay
fname.lname@inria.fr

Dan Vodislav
CNAM/CEDRIC
lname@cnam.fr

Abstract

Scalability is a key issue in current content distribution systems for web communities, which have to disseminate increasingly large amounts of data to increasingly large communities of users. We believe that peer-to-peer (P2P) architectures provide good solutions for scalable distribution of large amounts of data over the web. This paper presents EDOS, a P2P system for open-source software distribution, able to disseminate gigabytes of data to thousands of users on the web. We focus on the publishing and querying functionalities and we present several experiments realized on the Grid'5000 network, that confirm the system's scalability.

Keywords: peer-to-peer, content distribution, scalability, open-source software

1 Introduction

The growing importance of web communities, whose number and size augment every day, produced an increasing need of sharing, retrieving and transferring data on the web. In the particular case of open-source software (e.g. Linux), very large amounts of data (Gigabytes and more of source and binary code) must be disseminated to a very large community of developers and users (thousands and more). Moreover, content is frequently updated to new versions of the software modules.

For Linux, content is generally disseminated either as large ISO images of a full Linux release, or as a set of packages that group binaries or source code for a single software module. In the first case, content distribution is easier to manage, but a part of the content is uselessly transferred in the network at each download, because successive Linux releases have many common parts. On the other hand, the finer granularity in the second approach requires more complex data management, with frequent package updates inducing freshness problems.

Current software distribution architectures are mainly centralized or based on hierarchy of mirrors. While centralized systems are not scalable, current mirror-based systems face difficulties to guarantee reasonably uniform performances, because the effort is not uniformly shared between mirrors for each system functionality. Another major problem of such systems is the difficulty to globally ensure the freshness of content in the system.

More generally, current software distribution systems fail to fulfill several key requirements for a software distribution system, such as:

- avoiding excessive loads on the distribution servers and in the communication network, that lead to poor global performances;
- providing advanced content search based on metadata properties;
- providing support for maintaining freshness of content;
- guaranteeing robustness in case of failure of system components, the consistency of information and in general, some desirable QoS;
- being scalable with respect to the amount of data to be distributed and to the size of the network (number of users) for each system

functionality: publishing, querying, downloading, change notification, etc.

We believe that *peer-to-peer architectures* (P2P), that uniformly share the effort among participants and provide replication, are a good solution for scalable software distribution. Instead of addressing individual servers, users address the system as a whole, through one of its peers. Content distribution functionalities are realized through collaborative processes between peers, scalability being ensured by resource sharing: CPU, disk space, bandwidth, etc.

Motivated by the belief that incremental improvement of existing distribution systems will fail to fulfill the needs, we introduce a novel solution based on a P2P architecture. It has been developed in the context of the EDOS European project [1]. EDOS stands for *Environment for the development and Distribution of Open Source software* and addresses the production, management and distribution of open source software packages. The EDOS distribution system proposes a P2P dissemination architecture including all the participants to the distribution process: publishers, mirrors and end-users.

Compared to existing software distribution systems, EDOS introduces several key improvements:

- a P2P architecture providing resource sharing, load balancing and robustness;
- advanced information system capabilities, based on distributed indexing and querying of XML content metadata;
- efficient dissemination based on clustering of packages and multicast techniques;
- support for freshness maintaining on updates, based on a pub/sub mechanism.

This paper presents the EDOS distribution system and focuses on scalability evaluation issues for two main functionalities: publishing and querying. We report here about the first experiments on publishing and querying content metadata in EDOS and show that the system is scalable with respect to the amount of published metadata and to the size of the indexing network. A detailed description of the system may be found in [10].

2 Related work

Among the current distribution architectures for Linux, the most popular open-source software, only few of the various dissemination methods [9] propose improvements to the classical architecture. The most noticeable are Red Hat Network [15], that adds notification channels to maintain freshness, and Conary [8], that uses distributed versioning repositories to minimize downloads for updates.

Currently, P2P content distribution mainly addresses load balancing and bandwidth sharing (Coral [11], Codeen [6]). We extend this primary use by adding a *distributed information system* based on XML metadata indexing and querying, together with efficient *file sharing* and *multicast dissemination*, based on BitTorrent [7]. Among the various P2P infrastructures [5], *structured overlay networks* provide better performance for locating and querying large quantities of data. We use FreePastry, an implementation of the Pastry [14] distributed hash table system.

The EDOS P2P architecture is an example of *data ring* [4] structure for community content sharing. A data ring is a distributed system that is responsible as a whole for all the content sharing functionalities: indexing, replication, reorganization, etc.

3 System overview

The goal of the EDOS distribution system is to efficiently disseminate open-source software (referred at a more general level as *data* or *content*) through the Internet. Published by a main server, data is disseminated in the network to other computers (mirrors, end-users), that get copies of the published content.

EDOS is articulated around a distributed, P2P information system that stores and indexes *content metadata*. This metadata-based information system allows querying and locating data in the EDOS network.

3.1 Data model

There are *three kinds* of data units in the EDOS distribution system:

- **Package**: main data unit type, represented by an RPM file;

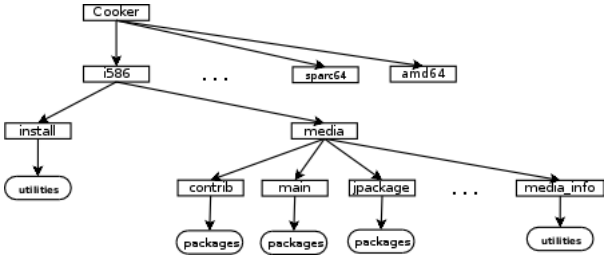


Figure 1: Data units for EDOS distribution

- **Utility**: individual file used in the installation process;
- **Collection**: it groups packages, utilities or subcollections, to form a hierarchical organization of data.

A release is a set of data units that form a complete software solution - it corresponds to a full Linux distribution. Its content is described by a collection.

Content dissemination is initiated by *publishing* data units in the system. Publishing consists in generating metadata for each data unit and indexing them in the distributed system. Periodically, the main server publishes a new release. Updates to the current release are realized by publishing new versions of packages or utilities. When the time comes, a "publisher" decides to transform the current status of the current release into a new release.

Figure 1 illustrates the organization of content in EDOS. We adopted a standard hierarchical organization of data. Such an organization is, for instance, used in the Mandriva Cooker distribution, where packages and utility files are grouped in collections at several levels, providing various levels of data granularity.

The example in Figure 1 corresponds to a release, whose root collection is *Cooker*, containing sub-collections *i586, ..., sparc64, amd64*, each one of them containing other sub-collections, packages or utilities.

Metadata management

This is a key issue in the distribution process. We built a global, distributed information system about data to be disseminated in the network. This system is fed with content metadata, that may include not only content properties, but also information on production, testing, statistics, etc. Distributed management of metadata is justified by its size and by the high rate of queries and updates it supports. The ability to express complex queries over metadata and to provide effective distributed management is a major contribution of the EDOS system.

In the largest sense, metadata consists in the set of properties that characterize data units. We classify metadata properties in three main categories:

- *identifiers* - in our case, the name and the version number uniquely identify a data unit.
- *static properties*, that do not change in time for a content unit, e.g. size, category, checksum, license, etc.
- *changing properties*, i.e. properties that may vary in time: *locations* of replicas in the network, *composition* for collections, statistics for the distribution process (e.g. the number of downloads of a package), etc.

The XML structure chosen for EDOS metadata is a compromise between efficiency requirements for both *query processing* (that requires large XML files, containing all the elements addressed in a query) and *metadata updates* (that need small files). Our choice was to create one separate XML file for each package, describing package properties, and one metadata file for each release, describing the release composition and utility files. For efficiency reasons, replica location management uses a separate mechanism.

More details about representing, publishing and querying EDOS metadata are presented in Section 5.

3.2 Actors and roles

Peers of the EDOS P2P distribution system may be classified in *three categories*:

1. **Publishers**: They publish new content in the network, manage flash-crowd dissemination and the pub/sub system.
2. **End-users**(Clients): They download content from other peers, query the system, subscribe to data changes. They also participate to the network by storing and providing portion of the data for downloads. To query the metadata they need an entry-point to the indexing network of the Mirrors.

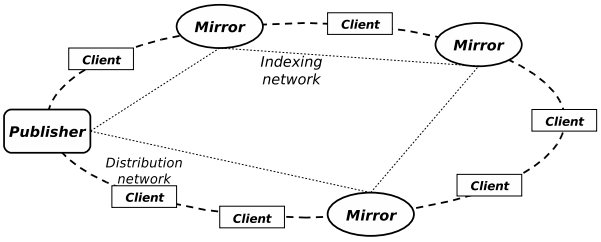


Figure 2: Actors in EDOS P2P content distribution

3. **Mirrors**: They provide all the functionalities of the End-users. Besides that, they participate in the indexing network. Typically, these are *trusted*[1] and reliable servers providing some guaranteed quality of service.

Figure 2 presents the actors in the P2P distribution network. Actors are connected in *two distinct networks*:

- *The distribution network*, composed of all the peers - they store, download and share EDOS data, i.e. software packages, utilities and collections.
- *The indexing network*, composed of trusted peers (Publisher and Mirrors) - they store the index on content metadata. For security reasons, Clients are not allowed to participate in metadata and index sharing, but can provide content, whose validity may be verified by using the checksum metadata property.

3.3 Usage scenarios

Flash-crowd situations

These generally happen when new, popular and large size content is published (here: a new release), and many peers (Clients or Mirrors) want to get this content as soon as possible. Flash-crowd distribution uses efficient dissemination methods, based on clustering of data units and multicast. Each peer asking for some portion of the new release may already have some of the packages - therefore it computes a *wish list* containing only the missing data units. Based on the wish lists gathered from peers, the Publisher computes *clusters* of data units to be disseminated. Instead of downloading individual data units, peers download clusters (where a cluster is a set of packages that are requested by a common set of peers), in a global *multicast* process.

The flash-crowd dissemination of a new published release is described by the following steps:

1. Peers interested in the new release subscribe to a special channel for new release publication.
2. The Publisher publishes the new release and notifies all the peers that subscribed to that release. Among the metadata published for the new release, its *composition* (identifiers of data units) is necessary for each peer to determine the set of data units to download.
3. Notified peers decide if they are finally interested in the new release or not. The peers compute the delta between the new release and the content they already have. This delta, called *wish list*, composed of the identifiers of data units to be downloaded, is sent to the Publisher.
4. The Publisher waits for wish lists during a predefined window of time. Then it computes clusters of data units, based on the set of collected wish lists.
5. The Publisher published the computed clusters of data units and starts "torrents" for disseminating them.
6. Each peer gets in parallel (via multicast techniques) a set of clusters that covers its wish list.

Off-peak distribution and query

This corresponds to periods between flash-crowd situations. During these periods, the Publisher may publish updates to the current release and the other peers may query the system, download query results, subscribe to distribution channels, receive notifications on such channels and download software updates.

4 Software architecture

EDOS distribution functionalities have been implemented as a *Java API*, based on a set of external software modules (Figure 3):

[1]The correctness of a file is guaranteed by its signature and checked at download. The correctness of metadata is guaranteed by the fact that the mirrors are trusted peers.

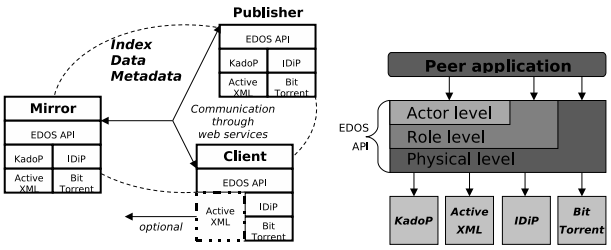


Figure 3: EDOS software modules and API structure

- **ActiveXML** [2] provides an extended XML format for EDOS metadata and storage for metadata documents published in KadoP. Web service calls embedded in ActiveXML documents may be used to represent intensionally changing information (e.g. statistics on the distribution process) or package dependencies.

- **KadoP** [3] is a very efficient distributed index for (Active)XML documents, that allows publishing, indexing and advanced querying of EDOS metadata. KadoP is the core of the EDOS information system.

  Based on the Pastry [14] distributed hash table (DHT), KadoP decomposes ActiveXML documents into key-value pairs stored in the DHT, and uses the DHT to evaluate XML queries over the metadata.

- **IDiP** [13] implements functionalities for the flash-crowd usage scenario: content clustering and multicast dissemination using BitTorrent.

- **BitTorrent** [7] is an efficient file sharing and downloading system. We use a slightly modified version of *Azureus*, a Java implementation of BitTorrent, for multicasting and downloading from multiple replicas.

The structure of the EDOS distribution API is presented in Figure 3. The API is organized into three levels:

1. **Physical level**: provides EDOS peer basic functionalities. The physical level is composed of several modules: a *content manager*, an *index manager*, a *channel manager*, a *dissemination manager*, etc. Programming distribution applications at the physical level requires more effort, but offers the best flexibility.

2. **Role level**: built on top of the physical level, provides a default implementation for each role in the distribution network, i.e. publishing, downloading, replicating, querying, and subscribing.

3. **Actor level**: provides a default implementation for each actor kind (Publisher, Mirror or Client), by combining several roles.

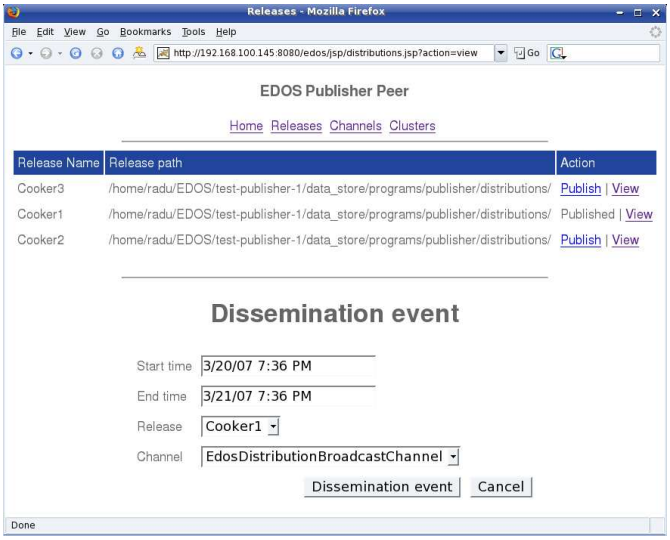


Figure 4: The Publisher GUI

The first version of the EDOS distribution system has been implemented as a set of Java/JSP web applications on top of the EDOS API. Each peer in the EDOS network runs a Java web application. Peer applications use a Tomcat web server for deployment, with Axis for web services. The functionalities of each EDOS peer are accessible through a JSP user interface, running in a web browser.

The Publisher web application (Figure 4) allows publishing new content, managing subscription channels and driving flash-crowd dissemination. Mirrors and Clients have the same user interface (Figure 5), allowing queries, downloading, subscriptions to channels and notification handling.

5 Publishing and querying EDOS metadata

We focus in this paper on *metadata-related functionalities* in EDOS: publishing and querying.

Figure 5: The Client/Mirror GUI

As explained above, publishing content in the network does not mean transferring content files over the network, but simply "announces" the system that new content is available and can be downloaded. This announce is realized by publishing metadata for the new content in the distributed index. Possibly, metadata publication may trigger a flash-crowd process or notifications on channels that concern the new content. Metadata is published in the form of XML files, as detailed below.

To know if some content unit is available in the system, one should use queries over metadata, that includes content unit identifiers and other properties. The system provides a powerful XML query language and distributed query processing capabilities.

Both publishing and querying metadata are based on the KadoP distributed index. We introduced several optimizations to KadoP in order to improve metadata-related operations. The main goal of this paper is to evaluate the system's scalability for publishing and querying metadata, and to compare various optimizations. The experiments are realized in the conditions (amount of metadata, number of peers) that correspond to the EDOS application.

5.1 Metadata representation

XML representation of EDOS metadata respects the following rules:

- For each software *package*, the package identifier (name and version) and its static characteristics (size, license type, checksum, etc.) are grouped into an XML metadata file.
- For each *release*, the release composition and other release properties are grouped into a single XML metadata file. The release composition contains information about the hierarchy of collections, about the identifiers of packages in each collection and about utilities.

Ideally, for query processing, all EDOS metadata should be placed in a single XML file. This would enable, for any query on metadata, to avoid joins between metadata files, that are potentially costly in a P2P network. On the other side, new content published in the system would require updates to this huge file, that would be very costly. The choice made for EDOS (one file/package plus one file/release) is a compromise between the needs of query processing and of metadata update.

Since not all possible queries on metadata are equally useful, this choice is also based on a study of the most usual queries in EDOS. Two categories of queries seem to be the most common for open-source software users:

- research of package identifiers and other properties given some values for the package metadata properties
- queries on the composition of a release or of some of its collections

The choice made for slicing of EDOS metadata in files avoids joins for these very common queries.

More precisely, the structure of a package metadata file is given by the following DTD, where only one property (SUMMARY) beside the identifier is shown.

```
<!DOCTYPE packageMetadata [
<!ELEMENT packageMetadata (id, SUMMARY, ...)>
<!ELEMENT ID (NAME, cversion)>
<!ELEMENT cversion (VERSION, RELEASE, ARCH)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT SUMMARY (#PCDATA)>
...
]>
```

The structure of the release metadata is described by the following DTD, where only release composition is detailed.

```
<!DOCTYPE releaseMetadata [
<!ELEMENT releaseMetadata
    (NAME, collectionMetadata*, utilityId*, ...)>
<!ELEMENT collectionMetadata
    (NAME, PATH, composition)>
<!ELEMENT composition
    (packageId | utilityId | collectionMetadata)*>
<!ELEMENT packageId (NAME, PATH, cversion)>
<!ELEMENT utilityId (NAME, PATH, cversion)>
<!ELEMENT cversion (VERSION, RELEASE, ARCH)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT PATH (#PCDATA)>
...
]>
```

A release contains several thousands of packages, organized on 3-4 levels of collections. In the current system, a metadata file for a package is about 1-2 KB in size, while the release metadata has several hundreds of KB.

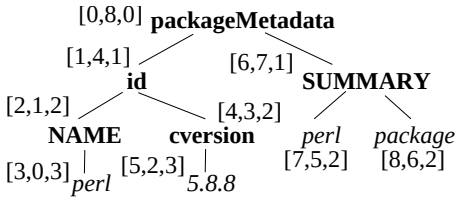
5.2 Massive publication

A key feature of the EDOS distribution system is the need for *massive publication* of metadata. Typically, when a new release has to be published, a large number of metadata files (thousands: one for each package in the release and for the release itself) must be published in a flurry in the distributed index.

In this context, publication is a time consuming process, that needs optimization. We introduce and test a simple but effective optimization technique for massive publication in KadoP, based on several Publishers that publish content metadata in parallel.

Publishing an XML file with KadoP [3] first consists of indexing its tree-like structure, such as in Figure 6. The document contributes to the index with a set of [*key, value*] couples, where the keys are *tags or words* in the document and the value for a key is *a list of node identifiers*. A node identifier contains the triple [*prefix, postfix, level*] that localizes the tag/word in the document tree and that is used for query processing, but also the document identifier in the network (not shown in Figure 6).

The global KadoP index is composed of the key-based fusion of all the key-value couples for all the documents in the network. In other words, for a given tag/word, the global index contains



packageMetadata	→	[0,8,0]
id	→	[1,4,1]
NAME	→	[2,1,2]
cversion	→	[4,3,2]
SUMMARY	→	[6,7,1]
perl	→	[3,0,3],[7,5,2]
5.8.8	→	[5,2,3]
package	→	[8,6,2]

Figure 6: Indexing a metadata file

the list of all the node identifiers in the network, labeled with that name. KadoP uses the Pastry distributed hash table (DHT) to store the global index, by uniformly distributing the key-value couples on all the peers in the indexing network. More details on the KadoP index are given in [3].

The massive publication optimization evaluated here consists of *distributing publication* by using several Publishers that publish content in parallel. In EDOS, the responsibility for publishing new content belongs to a single authority, the main software packager. Therefore, additional Publishers must be under the control of the main server. This means that each new release to be published must be split in several parts and distributed among additional Publishers. They will publish their part in parallel, under the control of the main server. A consequence is that the Publisher application becomes more complex, by adding functionalities for coherent distributed publishing.

On the other side, these peers must belong to the EDOS indexing network. We choose to build them as a *special kind of Mirrors*, enriched with publishing functionalities. This is realized by creating (through the EDOS API) a peer that includes all the functionalities of a Mirror *and* a publishing role.

We wish to evaluate the time necessary for publishing metadata in the EDOS system and verify its scalability, based on three parameters:

- The amount of published metadata.
- The size of the network, i.e. the number of

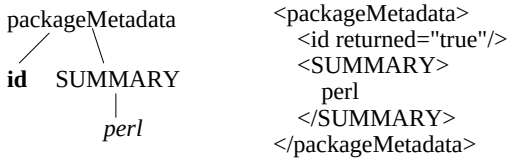


Figure 7: Query on metadata

peers in the indexing network.

- The number of Publishers that publish metadata in parallel

5.3 Query processing

KadoP provides a powerful query language for XML data, based on tree pattern queries. An example of KadoP query is presented in Figure 7, that shows both the tree form and its equivalent textual representation used by KadoP.

The meaning of the KadoP tree query is the usual one: the query tree is used as a pattern to filter documents that respect the structural relationships between elements and/or words. The default structural relation between a query node and its parent in the tree is "descendant" ("//"). The query in Figure 7 asks for package identifiers (composed of a name, a version, etc.) for packages that contain the word "perl" in the SUMMARY property.

Query processing is distributed among the peers in the indexing network, but is driven by the peer that asks the query. It mainly consists of three steps, among which the first and the last one require network communication:

1. For each node in the query tree, the peer asks for the corresponding index entries from the distributed index in the network.

2. The lists of node identifiers for each key are combined in order to check the structural relationships - this produces a set of combinations of node identifiers that match the query tree.

3. For each combination of node identifiers of the projected elements, the corresponding elements are retrieved from their documents in the network.

We wish to evaluate the variation of the query execution time and verify its scalability, based on three parameters:

- The amount of indexed metadata, i.e. the size of the global index.

- The size of the network, i.e. the number of peers in the indexing network.

- The size of the query, i.e. the number of nodes in the query tree, which determines the number of index entries to be retrieved and processed.

Since the query processing time depends on the size of the result (affecting step 3 above), we only evaluate the time necessary for steps 1 and 2.

6 Evaluation

The evaluation of the EDOS distribution system in real conditions, over hundreds of Mirrors and thousands of Clients connected to the Internet, is a difficult task. The main problem, besides the availability of a large set of peers, is the heterogeneity of the firewall/NAT configurations that disturbe the communication between any couple of peers in the network.

In this context, we choose the Grid'5000 network [12] for scalability evaluation and we conceived test scenarios focused on each functionality of the system. Grid'5000 offers several thousands of peers, organized in several clusters, over a high-speed network. The access to each node in the network is exclusive, based on reservations, and the connections are reliable. Even if this framework simplifies many problems found in a real context (large bandwidth, reliability, no firewalls, homogeneous software environment), it allows a first evaluation of the EDOS distribution system functionalities at a large scale.

We present here a set of early results on the evaluation of the EDOS distribution system, concerning the scalability of the publishing and querying functionalities.

6.1 Grid'5000 framework and parameters

The Grid'5000 platform represents an appropriate test bed for our P2P architecture, offering the required range of nodes for the application deployment.

The Grid'5000 network gathers 9 sites geographically distributed in France featuring a total of 5000 CPUs. The main purpose of this platform is to serve as an experimental test bed allowing experiments in all the software layers between the

network protocols up to the applications. Each site represents a cluster grouping up to 500 nodes, linked together with a fast ethernet of 1-10 Gb/s.

As we described in Section 3, we distinguish between two distinct EDOS networks: for *content distribution* and for *metadata indexing*. For the publishing and querying functionalities only ***the indexing network*** is concerned. We mention that in the current mirroring architecture used by Mandriva, we count only up to 50 mirrors in the distribution network. That means that the number of Mirrors in the EDOS distribution network (i.e. the size of the indexing network) can be limited to several dozens of peers. Therefore, in our experiments we do not consider networks larger than 100 peers for publishing and querying metadata.

The first scalability parameter considered in our experiments is called **INS** (Indexing Network Size) and we chose three representative values of 10, 30, and 100 nodes in the network.

The data set selected for the experiments represents a complete Mandriva Linux distribution, counting around 5000 packages. We called this parameter **MDS** (MetaData Size) and we observed its variation from 0 to 5000 data units. The *Cooker2007* Linux distribution we considered corresponds to a total size of 5 GB of data (package files) and to around 7 MB of metadata.

Also, for the querying part we used queries with different sizes (i.e. number of nodes in the query tree), in order to evaluate the impact of a common query parameter (the query size) on the query processing time compared to the scalability parameters (INS and MDS).

For both publishing and querying, the evaluation program has to create the EDOS indexing network on the Grid'5000 peers, then to publish metadata. For query evaluation, the corresponding set of queries is launched once metadata is published.

The general algorithm for the evaluation scenario consist in the following main steps:

1. Deploy and run the Publisher peer
2. Deploy and run all Mirror peers
3. Publish the *Cooker2007* release
4. If query evaluation, launch queries
5. Collect results and stop all the peers

This algorithm is implemented in a shell script that automatically launches all the processes in the experiment. This script is run on the Publisher peer and it manages all the distant peers, representing the Mirrors. The deployment process performs an installation of each peer on a local machine, followed by the start of each application. This process corresponds to the *initialization* of the indexing network. The Publisher is the peer that initializes the network and each Mirror peer joins the indexing ring afterward (steps 1 and 2).

After that all the Mirrors joined the network, the Publisher launches the *publish* method for the test release (Cooker2007) and starts the time counter (step 3). Method invocation for publishing and querying are done by web service calls on the corresponding peers. Each peer executing a publishing or querying method measures the time necessary for the operation and writes it in a local log.

Finally, the manager script collects the log files and stops all the applications (step 5).

6.2 Publishing functionality

We evaluate the time needed for publishing the content of a complete release, considering the following parameters:

1. **INS:** {10, 30, 100} peers
2. **MDS:** up to 5000 packages (complete Mandriva Cooker2007 release)
3. **Optimization:** single Publisher vs. multiple Publishers

The complete process for publishing the metadata consists in several tasks for each package: metadata extraction from the data units (packages), XML metadata file parsing and building of the sets of key-value couples, packing the key-value couples into messages at the Pastry level and sending them in the indexing network. We did not include in the evaluation the metadata extraction step and we considered that metadata files for all the packages were generated a priori.

Experiments with publishing are presented in Figures 8 and 9. They correspond to two different cases:

- **Single Publisher** and INS={10, 30, 100} (Figure 8)

- **INS=30** and multiple Publishers (1, 2, 5) (Figure 9)

Figure 8 compares the publishing times obtained using a single Publisher. The corresponding functions are monotonically growing with the number of packages: $f_{INS=10}(MDS)$, $f_{INS=30}(MDS)$, and $f_{INS=100}(MDS)$.

We remark that for larger sizes of the indexing network, performances are better and the time growth is (almost) linear with the size of the data. Two opposite factors are important when the network size grows. The first one is the lookup time in the DHT, that grows, being proportional to $log(INS)$. On the other side, the local index on each peer becomes smaller, because the global index is shared among a larger number of peers. It appears, by comparing results for INS=30 and INS=100, that the first factor becomes less important than the second one when MDS grows - the global time for MDS=5000 is smaller for INS=100 than for INS=30, even if for small MDS they are almost the same (a little bit smaller for INS=30). This could be explained by the high network speed and the caching mechanisms used by Pastry, which makes lookup almost as fast for both INS. On the other side, KadoP uses a database for the local index, and the access time to this index depends on its size. It appears that in the case of a fast network, the times for a local index update on disk and for a network transfer become comparable.

The variation for INS=10 shows faster publishing for small values of MDS (because of the lookup factor), but significantly slower publishing starting with MDS=2500. It appears that the size of the local index on each Mirror becomes important enough to produce a larger index update time, probably related to the local index caching mechanism.

In conclusion, the publishing time is not really influenced by the network size, whose growth seems even to benefit to the system for high-speed networks. The time grows almost in a linear way with the size of the published metadata, with an additional growth brought by the access to the local index database.

Figure 9 presents the speed up obtained by applying the multiple publishers optimization: $f_{P=1}(INS=30,\ MDS)$, $f_{P=2}(INS=30,\ MDS)$, and $f_{P=5}(INS=30,\ MDS)$.

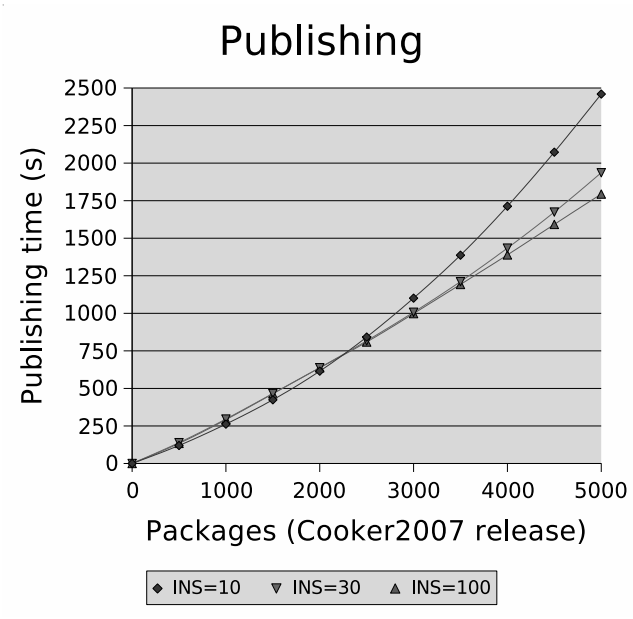


Figure 8: Publishing time for single Publisher and various indexing networks

Since the total number of packages is split in several parts processed in parallel, the processing time per Publisher is also divided by the same number. However, the speed up ratio is lower, because message processing in the indexing network introduces delays to the global publishing process. We remark that for two Publishers, the speed up variates between 1.5 and 2, while for five Publishers, the speed up is between 3 and 5. The explanation is that with more Publishers in parallel, the number of messages received by a peer grows and at some point the rhythm of reception may overrun the capacity of each peer to treat the incoming messages. This phenomenon is more visible for larger MDS, when the time for insertion in the local index grows.

We mention that the modular structure of the EDOS software API (see Section 4) enabled an easy extension of the Mirror application in order to add the publishing role to the Mirror peers.

In conclusion, distributed publishing is an effective optimization technique for massive publishing, that produces a significant speed up. Limitations occur when the number of Publishers grows too much, because of the limited capacity of a Mirror to treat messages for insertion into the local index. It is necessary that the number of Publishers remain small compared to the network size.

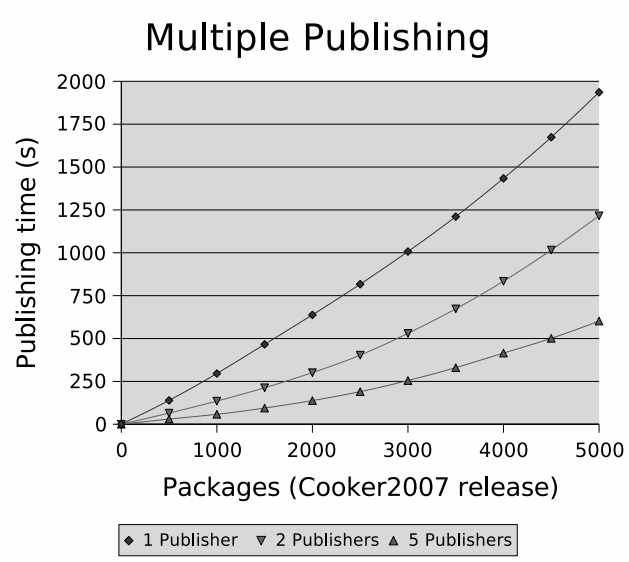


Figure 9: Publishing time for multiple Publishers on a 30 nodes indexing network

6.3 Querying functionality

We evaluate the querying performance of the system according to the following parameters:

1. **INS:** {10, 30, 100} peers
2. **MDS:** {1000, 2000, ..., 5000} packages
3. **Query:** three queries with different sizes {q0, q1, q2}

For the **size of the indexing network** we considered the same three instances of 10, 30, and 100 peers.

For each network, we publish the *Cooker2007* release, by stopping after each 1000 packages to query the published metadata. We obtained an evaluation of the query processing time for various query sizes, various MDS and various INS. The query processing time reported for each query is an average over 30 executions. As mentioned in Section 5, we do not include in the query processing time the time to get the XML results from each peer, in order to minimize the dependency on the result size.

We consider three queries of different sizes. The simplest one (q0) filters packages based on a single condition, the name of the packager:

```
<packageMetadata>
     <id returned='true'/>
     <PACKAGER>Olivier</PACKAGER>
</packageMetadata>
```

The second one (q1) adds some new query conditions and has a bigger size. We choose to only add conditions that do not change the number of results, i.e. conditions fulfilled by all the packages (e.g. the operating system to be "linux" or to have an URL tag in metadata). This way, the differences in execution time between q0 and q1 are only given by the size of the query and not by the size of the result. Query q1 is the following one:

```
<packageMetadata>
    <id returned='true'/>
    <DISTRIBUTION>Mandriva</DISTRIBUTION>
    <OS>linux</OS>
    <PACKAGER>Olivier</PACKAGER>
    <PAYLOADFORMAT>cpio</PAYLOADFORMAT>
    <PAYLOADCOMPRESSOR>gzip</PAYLOADCOMPRESSOR>
    <URL/>
    <VENDOR>Mandriva</VENDOR>
</packageMetadata>
```

The most complex query (q2) contains the complete list of metadata tags describing a package as well as some words, common to all packages ("linux", "cpio", "Mandriva"). The query has the following form, where the missing tags are present without any word:

```
<packageMetadata>
     <id returned='true'/>
     <DESCRIPTION/>
     <DISTRIBUTION>Mandriva</DISTRIBUTION>
     <OS>linux</OS>
     <PACKAGER>Olivier</PACKAGER>
     <PAYLOADFORMAT>cpio</PAYLOADFORMAT>
     <SUMMARY/>
     <SIZE/>
     <URL/>
     <VENDOR>Mandriva</VENDOR>
     ...
</packageMetadata>
```

Experiments with querying are presented in Figures 10 and 11. They correspond to two different cases:

- **Query=q1** & INS={10, 30, 100} (Figure 10)
- **INS=30** & Query={q0, q1, q2} (Figure 11)

In the first case (Figure 10) we measured the variation of the query processing time for the following cases: f_{q1}*(INS=10, MDS)*, f_{q1}*(INS=30, MDS)*, and f_{q1}*(INS=100, MDS)*.

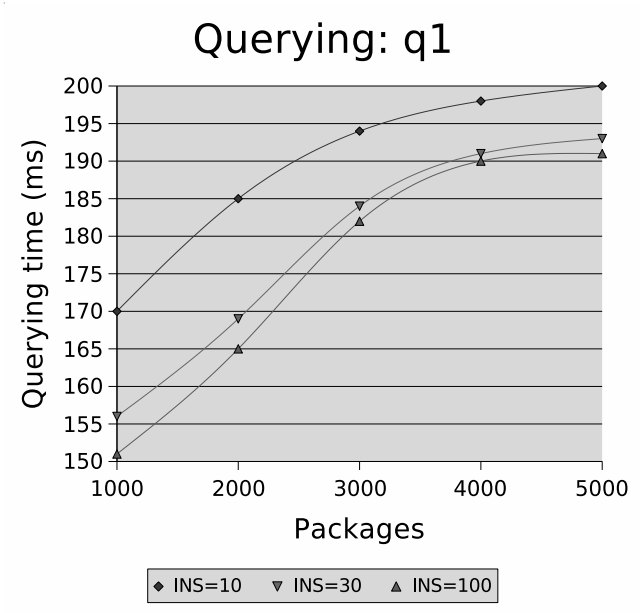


Figure 10: Querying time for various indexing networks

We remark that the query time grows with the size of the metadata for all network sizes. However, this growth is very slow, around 25% when MDS grows from 1000 to 5000.

We also remark that increasing the size of the network produces faster queries. The phenomenon is similar to the influence of the INS parameter on the publishing time. For querying also, the influence of the local index size is higher that the network's one. For larger INS, the lookup time grows (but only in a very limited way), while the size of each local index is smaller and the reducing of the access time to local indexes becomes more important than the growth of the lookup time.

In conclusion, query processing is scalable with the size of the published metadata (with a very slow growth) and even decreases when the number of peers grows (for high-speed networks).

In the second case (Figure 11) we show the evolutions of the querying time for various sizes of the query (*q0, q1*, and *q2*), considering the same configuration of the indexing network (*INS=30*). The graphs correspond to the functions: $f_{INS=30}$*(q0, MDS)*, $f_{INS=30}$*(q1, MDS)*, and $f_{INS=30}$*(q2, MDS)*.

We remark the same slowly increasing time for growing sizes of published metadata (MDS), confirmed here also for queries q0 and q2. We also

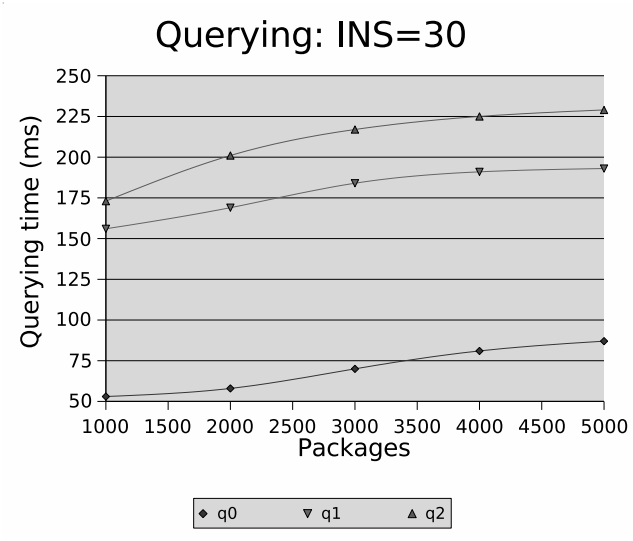


Figure 11: Querying time for various query complexities

remark that the size of the query has a clear influence on the processing time. The salient point is that the variation of the query size has an impact on the execution time that is comparable to (and even greater than) the growth of the published metadata size or of the network size.

In conclusion, query processing in EDOS is scalable. Moreover, the influence of the scale-related parameters (INS and MDS) is weak and comparable with the influence of usual query characteristics (the query size).

7 Conclusion

We presented in this paper a content distribution system and we described the functional and software architectures of the P2P information system. Metadata-related functionalities were analyzed in more details, with a focus on publishing and querying. We showed the scalability of the system in several experiments that we realized on the Grid'5000 network and we studied possible optimizations. A thorough analyze of the experiments is considered as future work, as well as the evaluation of the download and subscription-related functionalities of the system.

8 Acknowledgments

Itay Dar from Tel Aviv University for IDiP development and integration.

Nicoleta Preda from Gemo group for KadoP development.

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners (see https://www.grid5000.fr)

References

[1] S. Abiteboul, R. Pop et al. EDOS: Environment for the Development and Distribution of Open Source Software. In 1st International Conference on Open Source Systems, 2005. http://www.edos-project.org.

[2] S. Abiteboul, O. Benjelloun, B. Cautis, I. Manolescu, T. Milo, and N. Preda. Lazy Query Evaluation for Active XML. In *SIGMOD*, 2004.

[3] S. Abiteboul, I. Manolescu, and N. Preda. Constructing and Querying Peer-to-Peer Warehouses of XML Resources. In *ICDE*, 2005.

[4] S. Abiteboul and N. Polyzotis. The Data Ring: Community Content Sharing. In *Conference on Innovative Data Systems Research (CIDR)*, pages 154–163, 2007.

[5] S. Androutsellis-Theotokis and D. Spinellis. A Survey of Peer-to-Peer Content Distribution Technologies. In *ACM Computing Surveys*, 2004.

[6] Codeen. http://codeen.cs.princeton.edu.

[7] B. Cohen. Incentives Build Robustness in BitTorrent. In *Workshop on Economics of P2P Systems*, 2003.

[8] Conary Software Provisioning System. http://wiki.rpath.com/wiki/Conary.

[9] EDOS deliverable 4.1: Distribution of code and binaries over the Internet, 2005. http://www.edos-project.org/xwiki/bin/view/Main/D4-1/edos-d4.1.pdf.

[10] EDOS deliverable 4.2.2: Report on the P2P dissemination system, 2006. http://www.edos-project.org/xwiki/bin/view/Main/D4-2-2/edos-d4.2.2.pdf.

[11] M. Freedman, E. Freudenthal, and D. Mazieres. Democratizing Content Publication with Coral. In *1st USENIX/ACM Symposium on Networked Systems Design and Implementation*, 2004.

[12] GRID'5000 Plate-forme de recherche exprimentale en informatique, 2003. http://www-sop.inria.fr/aci/grid/public/Library/rapport-grid5000-V3.pdf.

[13] T. Milo and T. Zur. Boosting Topic-Based Publish-Subscribe Systems with Dynamic Clustering. In *SIGMOD*, 2007.

[14] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM Middleware*, 2001.

[15] S. Witty. Best Practices for Deploying and Managing Linux with RedHat Network.