Large Scale P2P Distribution of Open-Source Software

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

Itay Dar
Tel Aviv University, Israel
daritay@post.tau.ac.il

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

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

Dan Vodislav
CEDRIC-CNAM Paris, France
vodislav@cnam.fr

Nicoleta Preda
INRIA-Orsay, France
fname.lname@inria.fr

ABSTRACT

Open-source software communities currently face an increasing complexity in managing and distributing software content among their developers and contributors. This is mainly due to the continuously growing size of the software, of the community, the high frequency of updates, and the heterogeneity of the participants. We propose a large scale P2P distribution system that tackles two main issues in software content management: efficient content dissemination and advanced information system capabilities.

1. INTRODUCTION

In the context of an increasing need of sharing, retrieving and loading data on the web, the problem of distributing content to large communities across the web has acquired a growing importance. In the particular case of open-source software distribution (e.g. Linux), very large amounts of data (tens of Gigabytes) must be disseminated to a very large community of developers and users (up to thousands of members). Moreover, content is frequently updated to new versions of the software modules. For a Linux distribution, content is generally disseminated either as ISO images of a full Linux release, or as packages that group binaries or source code for a single software module. The problem with the first approach is that successive Linux releases have many common parts that users will uselessly download several times. The finer granularity in the second approach requires more complex data management, with frequent package updates inducing freshness problems.

The main requirements for a software distribution system could be summarized in the following four points: (i) avoid excessive loads on the distribution servers and in the communication network, that lead to poor global performances; (ii) provide advanced content search based on metadata properties; (iii) provide support for maintaining freshness of content; (iv) guarantee robustness in case of failure of system components, the consistency of information and in general, some desirable QoS.

Current distribution architectures, centralized or based on hierarchy of mirrors, fail to fulfill these requirements. For Linux, the most popular open-source software, only few of the various dissemination methods [8] propose improvements to the classical architecture, e.g. Red Hat Network [13], that adds notification channels to maintain freshness, or Conary [7], that uses distributed versioning repositories to minimize downloads for updates.

Motivated by the belief that incremental improvement of existing distribution will fail to fulfill the needs, we introduce a novel solution based on a P2P architecture. It has been developed in the EDOS European project [1], where 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.

We believe that P2P architectures, that uniformly share the effort among participants and provide replication, are a good solution for scalable software distribution. Currently, P2P content distribution mainly addresses load balancing and bandwidth sharing (Coral [10], Codeen [5]). 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 [6]. Among the various P2P infrastructures [4], *structured overlay networks* provide better performance for locating and querying large quantities of data. We use FreePastry, an implementation of the Pastry [12] distributed hash table system.

The main contributions of the EDOS distribution system are: (i) a P2P architecture providing resource sharing, load balancing and robustness; (ii) advanced information system capabilities, based on distributed indexing of XML content metadata; (iii) efficient dissemination based on clustering of packages and multicast; (iv) support for freshness maintaining on updates, based on a pub/sub mechanism. A detailed description of the system may be found in [9].

2. 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

VLDB '07, September 23-28, 2007, Vienna, Austria.

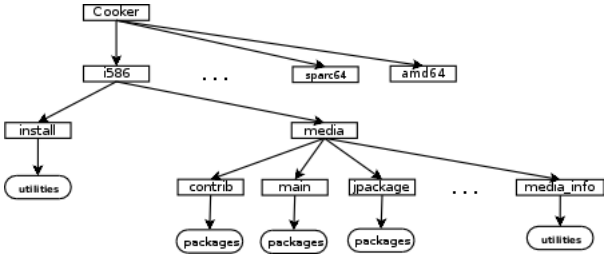

Figure 1: Data units for EDOS distribution

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.

2.1 Data model

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

- **Package**: main data unit type, represented by an RPM file;
- **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 into 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 subcollections *i586, ..., sparc64, amd64*, each one of them containing other subcollections, packages or utilities.

Metadata. management 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 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:

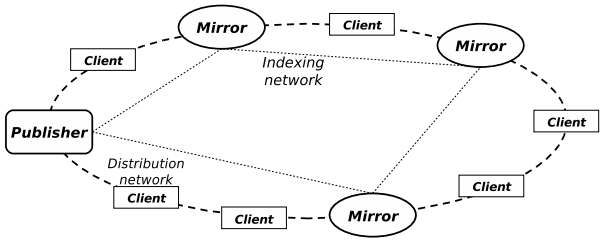

Figure 2: Actors in EDOS P2P content distribution

- *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: *location* of replicas in the network, collection *composition*, distribution statistics, 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 separate XML files for each package (package properties) and for each release (release composition). For efficiency reasons, replica location management uses a separate mechanism.

2.2 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 their local data for downloads. To query the metadata, they need an entry-point into the indexing network of the Mirrors.
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 automatically verified by using the checksum metadata property.

[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.

2.3 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 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.

3. SOFTWARE ARCHITECTURE

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

- **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.

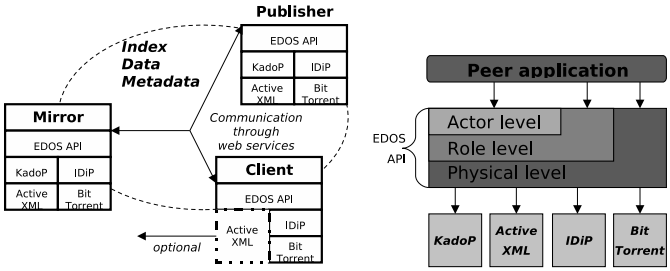

Figure 3: EDOS software modules & API structure

Based on the Pastry [12] 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** [11] implements functionalities for the flash-crowd usage scenario: content clustering and multicast dissemination using BitTorrent.
- **BitTorrent** [6] 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**: it provides EDOS peer basic functionalities. It 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**: it is 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**: it provides a default implementation for each actor kind (Publisher, Mirror or Client), by combining several roles.

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.

Tests with the first prototype demonstrated the relevance of P2P-based solutions for large-scale content distribution, the ability of managing very large amounts of metadata with KadoP and the improvements brought by IDiP for flash-crowd dissemination. More details are presented in [9]. We are now working on improvements in massive publication of metadata, in security (peer authentication), in firewall/NAT traversal and in the user interface. This various aspects will also be briefly demonstrated.

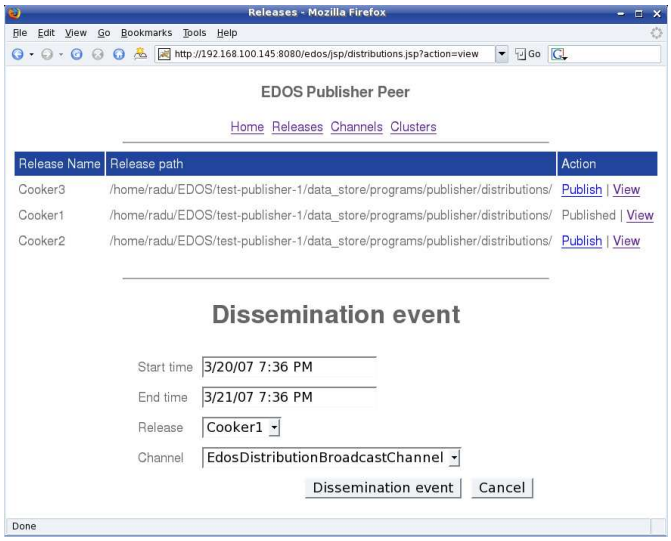

Figure 4: The Publisher GUI

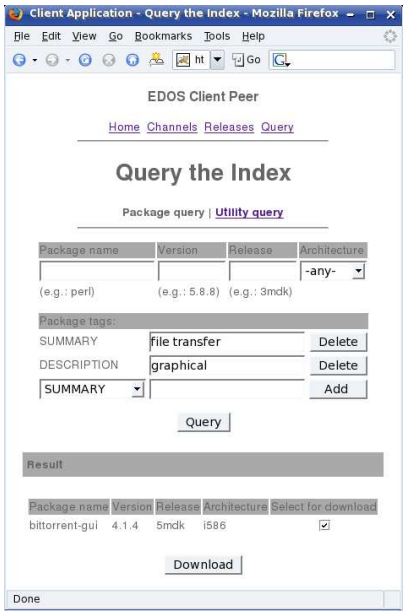

Figure 5: The Client/Mirror GUI

4. DEMONSTRATION SCENARIO

Our demonstration presents the functionalities of the system. We consider one Publisher and a set of Mirrors and Clients connected in the distribution network. Each peer runs a web application. The work of the peer can be controlled from a browser that provides the surveillance of the progress of its work. We consider three releases sharing common parts: one large release R_1, already published in the index and two other smaller releases, R_2 for demonstrating publication and flash-crowd dissemination, and R_3 for illustrating subscription/notification. Each peer is initialized with a different set of data units, which allows producing different wish lists towards flash-crowd dissemination.

The **publishing** functionality is illustrated in two steps. We first check that only release R_1 is published and visible on each peer. Then, on the Publisher we select and publish R_2 and we check on the other peers that R_2 appears in the index list.

The **downloading** functionality is illustrated in the *flash-crowd scenario*. A set of Client/Mirror peers select release R_2 for download. They compute and send to the Publisher their wish lists. The Publisher peer collects wish lists during the time-window, then it computes clusters of packages and creates torents used afterwards in the dissemination process (Figure 4). After the download is finished, we observe the effect of the updates.

The **querying** functionality is demonstrated on a Client peer, as shown in Figure 5. We use a query form to search for packages based on package identifiers and/or on a set of metadata tags. We show how a subset of packages from the result list can be selected and downloaded. A similar functionality is shown for querying utility files.

The **subscription/notification** functionality is illustrated by creating a subscription channel on the Publisher and by showing how Clients can subscribe to this channel. On the Publisher, we publish some new content (release R_3) and we show how some of the new packages are published in the new channel. We demonstrate how each subscriber receives notifications and an example of how they choose to handle it, e.g. by dowloading new packages.

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

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

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

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

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

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

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

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

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

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