A Hybrid Architecture for Content Consistency and Peer Synchronization in Cooperative P2P Environments

Carlo Mastroianni
ICAR-CNR
Via P. Bucci 41C
87036 Rende (CS) Italy
+39-0984-831725
mastroianni@icar.cnr.it

Giuseppe Pirrò
DEIS – University of Calabria
Via P. Bucci 41C
87036 Rende (CS) Italy
+39-0984-494773
gpirro@deis.unical.it

Domenico Talia
DEIS – University of Calabria
Via P. Bucci 41C
87036 Rende (CS) Italy
+39-0984-494726
talia@deis.unical.it

ABSTRACT

Peer-to-Peer architectures for content and knowledge management foster the creation of communities of workers in which effective knowledge and information sharing takes place. In such communities, workers have similar capabilities in providing other workers with data and/or services and are autonomous in managing their own knowledge objects. Since objects are typically shared among a set of workers, problems regarding concurrent access to content, content consistency and synchronization of peers arise. This paper describes a hybrid architecture for the management of data consistency and peer synchronization. The designed framework combines centralized, yet dynamic, mechanisms for metadata management and peer-to-peer mechanisms for data transfer. The paper reports on the use of these mechanisms in K-link+ a P2P collaborative platform, developed at the GridLab of the University of Calabria, for distributed knowledge management. An analytical study founded on queue network theory confirms the efficiency of the presented approach.

Keywords

Peer to Peer, Virtual Office, Collaborative Work, Distributed Knowledge Management

1. INTRODUCTION

Most of the current architectures for content sharing and knowledge management are typically based on client/server architectures in which one or more servers act as central entities. In these architectures, knowledge handled by workers must be managed according to organizational guidelines. However, these centralized approaches do not reflect the social nature of knowledge [1]. As discussed in [15], the *seed* of new knowledge is *individual* (*tacit*) knowledge, but its importance increases when it becomes available to the whole organization. Therefore, the *externalization* of tacit knowledge is a quintessential process to create new knowledge; this typically requires people to interact and collectively reflect on a problem or an idea. Such observations promote the demand for new technological architectures that place more emphasis on collaboration.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Infoscale '08, June 4-6, 2008, Vico Equense (NA), Italy.
Copyright 2008 ICST xxx-xxx-xx-xxxx-x

Moreover, since an increasingly number of workers very often operates outside of the traditional office, a virtual workplace where the physical workplace can be recreated is required. The *Virtual Office (VO)* solution complies with this requirement. A VO can be viewed as a work environment defined regardless of the geographic locality of the employees. This model is becoming essential since, even in conventional offices, today many business relationships are necessarily maintained across distributed environments. The VO is based on the concept of *workspace*. A workspace is a community of people that work together, as if they were in the same physical workplace, concurrently access shared content and accomplish common objectives. Such communities produce and exchange knowledge within workspaces through a set of shared tools.

The Virtual Office model cannot be effectively managed through a centralized entity in charge of object updating and dissemination, because this solution can hinder the autonomous interactions among workers and be poorly scalable. Conversely, the Peer-to-Peer (P2P) paradigm can be more appropriate and effective, because it fits both the requirements of collaboration (synchronous and asynchronous) and knowledge sharing that are raised by the adoption of VOs. In fact, P2P architectures naturally support the creation of communities (e.g., workspaces, peer groups) in which content and conveyed knowledge can be created, shared, exchanged and transformed.

We developed a P2P system, named K-link+ [13], which implements the VO model and allows users to create flexible and collaborative P2P applications for knowledge management. In K-link+, peers can concurrently work on the same shared documents/files, in the following referred to as "knowledge objects" or simply "objects". To foster peer autonomy, different local replicas of an object can be created, so concurrent access can affect data consistency if adequate mechanisms are not provided. Moreover, peers can join or leave the system at any time, thus introducing the *synchronization* issue: synchronization is required by peers that reconnect to the network and need to be informed about recent updates made on objects by other peers.

Object *consistency* is also a fundamental reliability requirement for a P2P system. Even if it is not possible, or convenient, to guarantee that all users are provided identical object replicas all the time, mechanisms must be provided to make users work without any actual limitations [19].

This paper describes an architecture which is designed for the management of knowledge in small/medium enterprises and is actually adopted in the K-link+ system. This architecture adopts a hybrid model to cope with the content consistency and peer synchronization issues. It exploits the efficiency of *centralized* models but at the same time includes *decentralized* features,

which assure scalability properties when the system size increases. This is accomplished by using: (i) a unique and stable server to maintain a limited amount of metadata information about shared objects, (ii) a number of interchangeable servers that maintain and manage the *primary copies* of shared objects, and (iii) a pure decentralized mechanism that allows P2P nodes to effectively exchange up-to-date object replicas.

To this aim different roles can be assumed by K-link+ Nodes (KLNs). In particular, a *Rendezvous* node maintains a common view about shared objects and their state. A set of *Manager* nodes are in charge of receiving object update requests from *Worker* nodes and possibly authorizing them. Finally, *Broker* nodes are used to speed up the propagation of updated objects over the network. The work presented here extends and refines the preliminary architecture description given in [14] and provides an analytical study through which the described hybrid architecture was assessed and important performance indices, such as computation load and response time, were evaluated.

The remainder of this paper is organized as follows: Section 2 discusses related work; Section 3 presents the architecture of K-link+ and its implementation in JXTA; Section 4 describes the hybrid model that addresses data consistency and peer synchronization issues; Section 5 presents an analytical performance evaluation of this approach within the K-link+ system; finally, Section 6 concludes the paper.

2. RELATED WORK

Replication of content is an important issue in P2P systems, especially if these are devoted to collaborative knowledge management [2, 4, 5, 9]. Replication mechanisms are usually classified into *reactive* and *proactive* mechanisms [17]. In reactive replication, as objects are transferred from the home node to the requesting peer, intermediate nodes through which the data flows, determine independently whether or not to cache the content. Some researchers propose to cache pointers instead of real objects in order to yield better query search performance. In DiCAS [22], queries are forwarded to peers of a predefined group which passively cache the pointers in an unstructured P2P network. However, a large overhead is necessary to update the pointers when the object is moved or deleted, since the updated location information has to be flooded to the whole overlay network.

In proactive replication, content is pushed to selected peers by the node that stores the primary copy, in order to obtain better performance in terms of query latency, load balance etc. However, the cost of replicating objects to a large number of peers can be cumbersome in both disk space and bandwidth, particularly for systems that support applications with large objects (e.g., audio, video, software distribution). A replication strategy based on object popularity in unstructured P2P networks is explored in [5]. Nevertheless, this strategy does not reduce the worst-case search latency for all the objects.

The strategy adopted in this paper borrows characteristics from both reactive and proactive approaches. A push-based mechanism is initiated by a peer when it generates or receives an updated version of an object, since it forwards this object to other *workers*, in a P2P fashion. This approach assures a quick dissemination of objects to the members of a community but, owing to its decentralized and unstructured nature, cannot guarantee that every worker is given the updated version of every shared object all the time. However, the updated version of an object is always maintained by the related *Manager* node. Therefore, when a worker cannot obtain the updated version of an object through the P2P mechanism, it can always request this object, with a *pull* modality, to the *Manager*.

An issue strictly related to replication is content consistency, which is a fundamental reliability requirement in distributed database systems, since concurrent operations on multiple replicas of the same data item can create a conflict [18]. This issue is even more complex in P2P systems, owing to their dynamic and unreliable nature. Current approaches differ according to the scale of P2P systems. In a large-scale and dynamic system, it is complex and cumbersome to guarantee full consistency among replicas, so researchers have designed algorithms to support consistency in a best-effort way. In [8], a hybrid push/pull algorithm is used to propagate updates, where flooding is substituted by rumor spreading to reduce communication overhead. SCOPE [3] is a P2P system that supports consistency among a large number of replicas, at the cost of maintaining a sophisticated data structure. By building a replica-partition-tree (RPT) for each key, SCOPE keeps track of the locations of replicas and then propagates update notifications.

Conversely, in a small- or medium-scale system, it is possible to adopt centralized schemes to guarantee a strong consistency model, which is often the *sequential* model [12]. In [21], an algorithm for file consistency maintenance through virtual servers in unstructured and decentralized P2P systems is proposed. Consistency of each dynamic file is maintained by a virtual server (VS). A file update can only be accepted through the VS to ensure the one-copy serializability.

The hybrid architecture described in this paper, and adopted in the K-link+ system, is specifically designed for knowledge management in small/medium enterprises. Its main purpose is to combine the efficiency of centralized models and the scalability and fault-tolerance characteristics of decentralized systems.

3. ARCHITECTURE AND IMPLEMENTATION OF K-LINK+

K-link+ is a collaborative P2P system that provides users with a Virtual Office environment, in which content can be shared, to enable collaborative work, and replicated to foster peer autonomy [13]. Different applications (document sharing, messaging, shared boards, agenda, etc.) can be integrated within a single environment (a K-link+ *workspace*) and new tools can be added as new components to support emerging requirements. In this section, the system architecture is briefly presented along with its implementation in JXTA [20]. For a more detailed description of the K-link+ architecture refer to the GridLab website, http://grid.deis.unical.it/k-link.

3.1 The K-link+ Architecture

The K-link+ architecture is based on five layers including basic grouping and communication services, data handling services, semantics services, workspace management services and a set of high level tools for content sharing and user cooperation. Figure 1 shows the K-link+ architecture, whose layers are described in the following.

K-link+ Core Layer. This layer defines the K-link+ core services that are exploited by higher layers. In the current version of K-link+, we adopted the JXTA framework to implement these services, though any other P2P infrastructure can be used. Among the core services, the *K-Group Service* allows KLNs to create new *K-Groups* (e.g., communities or workspaces), while the *Connection Service* allows KLNs to join the K-link+ network at

any time. Features used to send and receive messages are provided by the *Communication Service.*

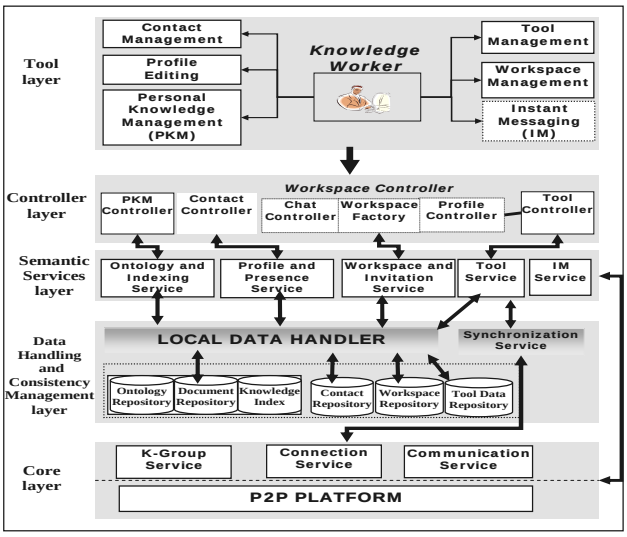

Fig. 1. The K-link+ architecture.

Data Handling and Consistency Management Layer. This layer is responsible for handling the problems introduced in the introductory section, that is, concurrent access to shared objects, object consistency and peer synchronization. The layer includes the *Local Data Handler*, which manages a set of local repositories to store information about contacts, workspaces, objects and so on. A detailed description of the functionalities of this layer is provided in Section 4.

K-link+ Semantic Services Layer. The services of this layer manage local and remote operations performed by a K-link+ user. The *Ontology and Indexing Service* deals with operations involving ontologies (creation, update) that K-link+ exploits to describe and annotate resources in a semantic way. The *Indexing Service* copes with the indexing of documents for keyword based searches. The *Profile and Presence Service* manages state check operations and enables users to create and publish their profile within the K-link+ network. The *Workspace and Invitation Service* handles workspace set up and population by sending invitation messages to other KLNs. The *Tool Service* is used to add new tool instances to workspaces at run time. The *Instant Message (IM) Service* allows KLNs to communicate each other via a chat-like system.

K-link+ Controller Layer. This layer contains a set of controllers through which the system interacts with the *Data Handling and Consistency Management Layer*. The *Workspace Controller* manages workspace settings through the creation of *workspace profiles* that contain information about workspace topics, sets of tools and workspace members. The *Contact Controller* enables workers to discover other peer workers in the network and add them to a personal *Contact List*. The *PKM Controller* is delegated to manage personal knowledge, that is, knowledge that is autonomously handled by workers. The *Tool Controller* allows workers to handle operations (add, update, remove) on knowledge management tools.

K-link+ Tool Layer. This layer provides a basic set of *K-link+ Tools* (file sharing, shared calendar, contact manager, etc.) that can be embedded in a workspace. In addition, other tools can be developed and included in the system as modular components, with the only requirement that the *K-link+ tool interface* must be implemented. Tools are shared among workspace participants: when a new tool is added to a workspace, a local tool instance will be automatically created on the K-link+ client of each workspace member. Hence, objects created through workspace tools can be shared among workspace members. For further information about the development of tools refer to the GridLab website, http://grid.deis.unical.it/k-link.

3.2 K-link+ implementation

While the K-link+ technology is independent of any particular P2P architecture, however the JXTA framework [20] was used to implement it.

As a consequence of this choice, the *Core Layer* of the K-link+ architecture maps each K-link+ peer, also called *K-link+ Node* (KLN), to a JXTA peer and each *K-link+ group* (K-group) to a JXTA peer group. JXTA peers can be divided into two categories: *edge peers* and *rendezvous peers*. Edge peers are transient peers provided with resource discovery and publishing capabilities. On the other hand, rendezvous peers are expected to be more stable. The aim of a rendezvous peer is to enable edge peers to discover and publish network resources. In JXTA each network resource (e.g., peers, peer groups, services) is published through XML documents called *Advertisements*. To participate in a P2P network, an edge peer needs to know how to connect to at least a rendezvous peer. Typically, an edge peer maintains a list of known rendezvous peers (called "seed" rendezvous) and participates in dynamic discovery of new rendezvous nodes. This allows edge peers to fail-over to an alternative rendezvous when needed, so as to enhance overall network reliability. Every JXTA peer can act both as an edge peer or a rendezvous peer. In fact, an edge peer can adaptively become a rendezvous peer if it cannot connect to any rendezvous for an extended period of time

4. CONTENT CONSISTENCY AND PEER SYNCHRONIZATION

In K-link+, several users can work concurrently on shared objects. To favor the autonomy of users, the system allows different replicas of the same object to be created, so that users can work on their local copies. As mentioned in Section 3.1, the purpose of the *Data Handling and Consistency Management* layer is to ensure data persistence, consistency management and peer synchronization. In the context of K-link+, the *sequential consistency* model is adopted [12], which assures that all updates performed on an object are seen in the same sequence by all the peers. The model is implemented by associating, to each object, a K-Link+ node (called *Manager*), which is responsible for authorizing object updates thus allowing the KLNs to view the updates in the same order. In particular, each object is assigned a *Version Number (VN)*, which is incremented after each update.

In more details, K-link+ defines the following set of *roles* that can be assumed by workspace nodes:

- *Creator*. It is a KLN that creates a shared object and specifies the *Manager List* (ML), i.e. the list of KLNs that can assume the *Manager* role for this object. Managers are ordered on the basis of their responsibilities in managing the object.
- *Rendezvous*. For each workspace, one rendezvous node maintains metadata about all the shared objects in a *Consistency Table* (described below) and provides such information to workspace members. The Rendezvous stores

up-to-date information about objects, in particular the identity of the node which is currently in charge of each object (i.e., the *Current Manager*) and the current VN.

- *Manager*. An object Manager is a KLN that manages the object life cycle and is contacted by KLNs when they want to propose an object update. An object can be assigned to several Managers, but at a given time only the *Current Manager*, i.e., the first online Manager in the ML, is actually responsible for the object. The Current Manager can decide whether or not to authorize an object update, according to the specific set of semantic rules associated to the object. KLNs are informed about the identity of the Current Manager by the Rendezvous.
- *Broker*. It is a KLN that maintains an updated copy of an object and can forward it to other KLNs. Whereas the Manager role is assigned at object creation time, the Broker role is *dynamic*, since it can be played by any node whenever it maintains an updated copy of an object.
- *Worker*. It is an ordinary KLN that operates on an object and possibly issues update proposals to the *Current Manager*. Workers can obtain an updated copy of an object either by a *Broker*, in a P2P fashion, or by the *Current Manager* of the object.

The *Rendezvous* maintains information about the state of the objects in a *Consistency Table*. Each object is permanently associated to an *Entry* of this table, whose structure is shown in Table 1. An object is identified by a unique *ID*, which is assigned when the object is created. Moreover, to keep trace of the object state, the Consistency Entry includes a version number VN (an integer value), which is incremented at each authorized object update, the *ID* of the Current Manager and the *Manager List*. While the *Rendezvous* is in charge of maintaining updated information about all the shared objects of the workspace, KLNs can maintain replicas of the Consistency Entries describing the objects in which they are interested.

Table 1. Consistency Table

Field	Description
Object ID	A unique ID, that identifies the shared object
Version Number (VN)	Object version number, incremented at each object update
Current Manage	The first online node in the Manager List. It is responsible for a shared object
Manager List	An ordered list of nodes that can assume the Current Manager role
Creator	The node that creates the object

The definition of the mentioned roles enables three different kinds of interactions, as shown in Figure 2. A *static centralized* approach is adopted when workers interact with the unique Rendezvous of the workspace. The presence of a single Rendezvous is appropriate in a small/medium network, as it is generally possible to assign this role to a node with high reliability features. Note, however, that the load of this node is moderate, as it only deals with small size metadata information, as is better discussed in Section 5. In fact, the aim of the Rendezvous is to provide reliable and updated information about objects, but the actual management of each single shared object is delegated to the corresponding Current Manager. This enables a *dynamic centralized* paradigm because the role of Current Manager, if and when needed, can be switched from one Manager to another that is included in the ML of the object. This way, several issues can be tackled: (i) the presence of a central bottleneck, which would be originated if all objects were managed by a single node, is avoided; (ii) it is possible to cope with the volatile nature of P2P networks, in which peers with Manager responsibilities can leave the network at any time; (iii) a Current Manager switch can be performed for an object also to better balance the load among different Managers, as will be described in Section 5.
On the other hand, a *decentralized* approach is exploited by Brokers that provide updated copies of objects to workers in a P2P fashion. The combined use of these three paradigms can represent an efficient trade-off among different ways to face distributed object management.

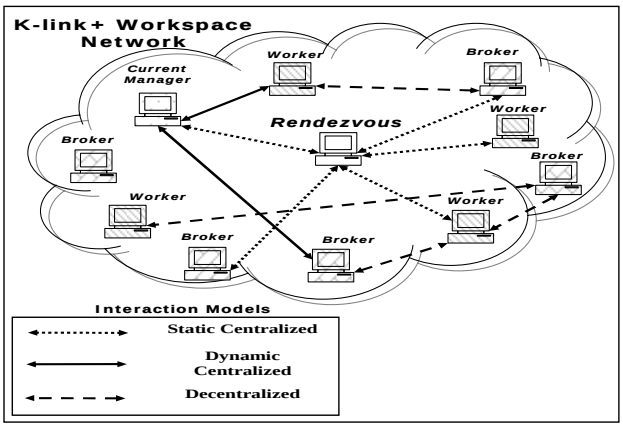

Fig. 2. The K-link+ approach to content consistency.

In the following different scenarios, in which the above-mentioned types of interactions occur, are described. These scenarios are: (a) the creation of a new shared object, (b) the update of an existing shared object, (c) the synchronization of a peer and (d) the Manager switch, performed either after a Manager disconnection or to achieve a fairer load balancing of Managers. Tables 2-5 list the various types of messages used in these scenarios. They are grouped by target node, as this will be useful for the evaluation of the computation load, done in Section 5.

Table 2. Messages received by the Rendezvous node

Message	Sender	Receiver
Create object	Creator	Inform the Rendezvous about the new object
Object information request	Worker	Check the current version number of an object
Manager leave	Current Manager	Inform the Rendezvous that the Current Manager is leaving the network
Version update	Current Manager	Inform the Rendezvous about the new version number of an object

Table 3. Messages received by Current Manager nodes

Message	Sender	Receiver
Online update request	Worker	Propose an object update while online
Offline update request	Worker	Propose an object update after reconnecting

Table 4. Messages received by Worker nodes

Message	Sender	Receiver
Object copy reply	Broker	Send an updated copy of an object
Version check	Broker	Ask a worker to check the version number of an object
Version information reply	Workspace Network	Inform the worker about the current version number of an object
Update reply	Current Manager	Accept/decline update proposals
Object information reply	Rendezvous	Give information about the current Manager and current VN of an object
Object copy request	Worker	Request an updated copy of an object

Table 5. Messages received by all Workspace nodes

Message	Sender	Receiver
Object forward	Creator	Forward a copy of a new object along with metadata to the interested peers
Version information request	Worker	Check if an updated copy of an object is available
Manager alive	Rendezvous	Inform the network about the identity of a new Current Manager

Scenario A. Creation of a new Shared Object

The creation of a new shared object is performed as follows. After creating a new shared object, a KLN (i.e., the Creator) informs the Rendezvous by sending it a *create object* message which contains metadata describing the new object (i.e., a new Consistency Entry), which will be stored in the Consistency Table. Moreover, the Creator defines the Manager List (ML): the first online Manager specified in the ML automatically assumes the role of Current Manager. The Creator forwards the new Consistency Entry, along with a copy of the new object, to the KLNs that can be interested in this object, by sending *object forward* messages. The KLNs store the received copy of the object in the local object repository through the *Local Data Handler*, while the Consistency Entry is stored into the local *Consistency Table*. When a KLN receives a new object it becomes a Broker, since it owns an object whose version number is the same as that maintained by the Rendezvous. A Broker can forward the new object to other KLNs in a P2P fashion, thus making object propagation faster.

Scenario B. Object Update

A worker can perform read operations, or provisional write operations, directly on the local copy of an object, through the *Local Data Handler*. However, every attempt to permanently modify the state of a shared object must be forwarded, through the *Synchronization Service*, to the Current Manager of the object, by sending it an *online update request* message. The Current Manager accepts modifications if these do not conflict with the current object state, according to the specific set of semantic rules associated to the object. If a modification is authorized, the Current Manager increments the object VN and sends back an *update reply* message to the requesting worker. Whenever an object update proposal is accepted, the updated copy of the object, along with information about the new VN, is sent from the requester to the involved workspace members, in a P2P fashion, through *object forward* messages, whereas the updated Consistency Entry is sent by the Current Manager to the Rendezvous through a *version update* message.

Notice that the propagation of the updated object is initiated by the requester instead of the Current Manager, thus avoiding to overload the latter. The KLNs that receive an updated object copy of an object assume the role of Broker for this object. To foster object propagation, a Broker may contact a set of workers by sending them a *version check* message containing the current object VN. If the worker notices that this VN is higher than that maintained locally, it replies to the Broker with an *object copy request* message and will receive the updated object copy through an *object copy reply* message. If the Current Manager is not available when an update request is issued, a *Manager Switch* procedure is required, as detailed in Scenario D.

Scenario C. Peer Synchronization

A synchronization procedure is performed when a KLN reconnects to the workspace network after being offline. Its purpose is: (i) to provide the reconnecting KLN with updated information about the objects of interest; (ii) to enable the KLN to propose possible object updates made on the local copy while offline.

In the first step, the KLN node uses the *Synchronization Service* to contact the Rendezvous and get information about current VNs and Current Managers of the objects of interest. This information is obtained by exchanging *object information request/reply* messages. Subsequently, two different procedures are followed by a KLN depending on whether or not it has performed any object update while offline. If no updates have been made, the *decentralized* approach (see Figure 2) can be exploited, since the KLN can obtain the latest object version from a workspace Broker. Specifically, the KLN checks whether the object VN received by the Rendezvous is higher than the VN stored locally, which would mean that the object has been updated. In this case, the KLN issues a *version information request* message to the workspace network and receives *version information reply* messages from workspace Brokers. Afterwards, the KLN chooses a Broker from which it can obtain the updated object in a P2P fashion, by using *object copy request/reply* messages.

A different procedure is followed if the KLN has made offline updates. In this case, the *dynamic centralized* approach must be adopted, since the KLN has to submit its update proposals to the Current Manager by sending to it *offline update request* and receiving by it *update reply* messages, following the same procedure described in Scenario B (*Object Update*).

In the case in which the Current Manager is not available when the KLN reconnects, a *Manager Switch* procedure is required, as detailed in Scenario D. In this case, the KLN keeps its update proposals stored in a local buffer until it is informed by the Rendezvous about the presence of an available Current Manager. In the meantime, the KLN can obtain an updated copy of the object from a Broker.

Scenario D. Manager Switch

In the above-mentioned scenarios, it is assumed that the Current Manager is online and available. If this condition does not hold, a *Manager Switch* procedure is required. By default the Current Manager is the first online KLN contained in the ML. When a new Manager becomes Current Manager, the Rendezvous informs the workspace network through a *manager alive* message. This way workspace members can store information about the new Current Manager (by updating the local Consistency Entry of the object) and will submit to it future update proposals.

However, in a P2P scenario, the Current Manager can leave the network at any time either in a *safe* or *unsafe* way. In the first case, it sends a *manager leave* message to the Rendezvous. The latter searches for the next online Manager contained in the ML and informs the workspace network through a *manager alive* message. If the Current Manager leaves the network abruptly (i.e., without informing the Rendezvous), a different approach is adopted. The Rendezvous is informed about the Current Manager failure directly by a worker. This can happen either when a worker reconnects (Scenario C) or when it receives no reply after an *online update request* (Scenario B). In both cases, the worker sends an *object information request* message to the Rendezvous. Before responding with an *object information reply* message, the Rendezvous always checks the availability of the Current Manager. If the Current Manager who is in charge of the object has left the system and another Current Manager can be elected, the Rendezvous operates the switch and informs both the requesting worker and the workspace network through a *manager alive* message.

5. SYSTEM EVALUATION

This section presents a performance evaluation of the proposed model for data consistency and peer synchronization. The main purpose of our performance analysis is to evaluate the load of Manager and Rendezvous nodes.

Analysis is made through a mathematical model based on the queuing theory and often adopted for the performance evaluation of computer systems [11]. Parameters adopted in the evaluation were experimentally determined during the actual operation of the K-Link+ platform in our departmental network. In particular, these parameters concern the size and frequency of client requests and the corresponding service times experienced on the Rendezvous and the Managers.

The arrival of messages and their processing is modelled through M/G/1 queues [10]. An M/G/1 queue consists of a FIFO buffer with requests arriving randomly according to a Poisson process at rate λ and a processor, called a server, which retrieves requests from the queue and serves them on a first-come-first-serve (FCFS) order, with a generic (G) distribution of service time.

The service time of requests is heavy-tailed in nature [6, 7]. In particular, the task size is often modelled with a Bounded Pareto distribution. According to this distribution, a high percentage of tasks require a short processing time, while a low percentage require long processing time. As opposed to the Pareto distribution, the Bounded Pareto distribution allows for the definition of minimum and maximum task sizes. This prevents the possibility of generating very long or very short tasks, which are not realistic. The probability density function for the Bounded Pareto $B(k,p,\alpha)$ is reported in formula (1).

$$f(x) = \frac{\alpha k^{\alpha}}{1-(k/p)^{\alpha}} x^{-\alpha-1} \quad (1)$$

In this formula, α represents the task size variation, k is the smallest task size and p is the largest task size. This function is defined for $k<=x<=p$ and expresses the distribution probability of the service time. Values of k and p were set according to measurements taken during the actual usage of K-Link+ at the GridLab of the University of Calabria and at the ICAR-CNR institute. The parameter α must be included in the range <0,2> (a lower value accounts for higher variability), and is set to 1 for our analysis.

The theory of M/G/1 queues enables the calculation of several interesting indices [16], that is, the average *load* on the server, the average *processing time*, needed to process a request at the server, the average *waiting time* of requests in the queue and the *overall service time*, which is the sum of the waiting time and the processing time at the server.

In particular, the average load, ρ, can be calculated as λ/μ, where λ is the average frequency of request arrivals at the server and μ is the inverse of the average processing time E(X), which can be calculated as the first moment of the Bounded Pareto service time distribution. The expected waiting time of a request in the queue, E(w), can be obtained by using the Pollaczek-Khintchine (PK) formula and the Little's law [10]. This results in formula (2), in which $E(X^2)$ is the second momentum of the Bounded Pareto distribution.

$$E(w) = \frac{\lambda E(X^2)}{2(1-\rho)} \quad (2)$$

The overall service time E(T) is simply obtained by adding to formula (2) the average processing time $E(X)=1/\mu$. The service time is only defined in the case that the average load is lower than 1, that is, if λ is lower than μ, otherwise the queue will grow indefinitely. Actually, the average load can be interpreted as the average CPU utilization needed to cope with the incoming messages. A value greater than 1 indicates that the node is overloaded and more servers are necessary to cope with the flow of requests.

In the next subsections, the performance of the most critical categories of nodes of the K-Link+ architecture are separately evaluated, that is, the Manager and the Rendezvous. To obtain λ, the arrival rates of the different types of requests/messages that are delivered to the Rendezvous and to the Managers are calculated and, according to the composition property of Poisson processes, these arrival rates are then summed.

5.1 Evaluating the Manager load

To estimate the load of a Manager network composed of up to 100 nodes and containing a number of shared objects ranging from 100 to 2000 are considered. Those values correspond to the objects on which clients are actually working. It means that there can be other shared objects but they do not concur to the system load if users are not working on them. In this sense, the maximum

number of objects (2000) corresponds to an average of 20 objects on which each client is actually working. Table 6 summarizes the parameters and values that have been adopted for our analysis.

Table 6. Scenario for the evaluation of the Manager load

Parameter Name	Value
Number of workers, *N*	100
Average fraction of online and offline nodes, *Fon* and *Foff*	0.5 and 0.5
Overall number of shared objects, *Nobj*	100 to 2000
Number of Manager nodes, *Nmg*	1 to 24
Average rate of operations that a worker performs on a shared object while online, *Ron*	1 each 6000 s
Average rate of operations that a worker performs on a shared object while offline, *Roff*	1 each 12000 s
Minimum task size	200 bytes
Maximum task size	100 Kbytes
Average time required by a Manager to process an update request , 1/μ	450 ms

In particular, the size of content that must processed by a Manager, when it evaluates an update request for an object, is comprised between 200 bytes and 100 Kbytes, which are the values experienced during K-link+ operation. The corresponding service times vary from 20 ms to 10 s: these were set as the values of parameters *k* and *p* in the Bounded Pareto distribution, reported in formula (1). The average service time, 1/μ, is obtained analytically, as the first moment of the Bounded Pareto distribution.

The load of a Manager node is computed as the contribution of two types of messages (see Table 3): *online update requests* incoming from online nodes, and *offline update requests* that are received from nodes that reconnect to the network. Actually, several offline requests can be sent by a node when reconnecting, so possibly generating a burst of requests. However, since these bursts come from different nodes at different times, their impact was found to be insignificant, so only the average arrival rates can be considered. The arrival rates corresponding to online and offline update requests, respectively named λon and λon, are calculated as follows:

$$\lambda_{on} = N \cdot N_{obj} \cdot R_{on} \cdot F_{on}$$

$$\lambda_{off} = N \cdot N_{obj} \cdot R_{off} \cdot F_{off} \qquad (3)$$

In the hypothesis that all the Managers receive comparable number of requests, the average arrival rate at a Manager, λ, is computed by dividing the sum of these 2 contributions by the number of Managers:

$$\lambda = \frac{\lambda_{on} + \lambda_{off}}{N_{mg}} \qquad (4)$$

From λ, performance indices can now be calculated as described in the previous subsection. Figure 3 depicts the Manager load ρ in a network with 100, 500, 1000 and 2000 objects, and different numbers of available Managers, in the hypothesis that the load is fairly shared among the Managers.

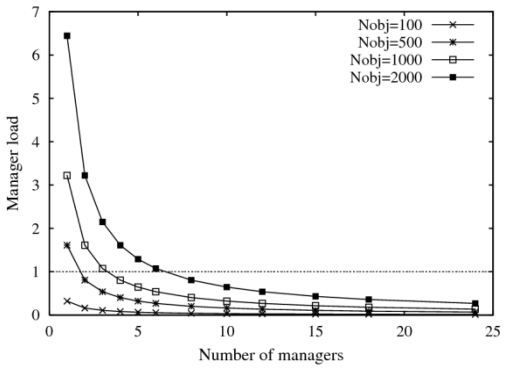

Fig. 3. Manager load vs. the number of Managers, for different numbers of shared objects. The load is sustainable only when it is below the dashed line, correspond to the value of 1.

The figure shows that the Manager load decreases with the number of Managers, with a negative exponential trend. It can also be noticed that, in the presence of a single Manager, the load is sustainable in the case of 100 shared objects, while the presence of more objects leads to a load greater than 1. For example, the load is about 1.65 if there are 500 shared objects, and is even higher with 1000 or 2000 shared objects. In these cases, a multiple Manager configuration is necessary, and the proper number of Managers can be chosen according to the number of objects. For instance, Figure 3 shows that at least 7 Managers are needed if the number of shared objects is 2000, since the load is always larger than 1 if fewer than 7 Managers are available.

Figure 4 shows the average overall service time *E(T)* (that is, the waiting time in the queue plus the actual processing time at the server), which is defined only when the corresponding Manager load (Figure 3) is lower than 1. It can be noticed that the overall service time tends to be very high as the corresponding value of the load approaches the value of 1, for example, in configurations with 1000 objects and 4 Managers or with 2000 objects and 8 Managers. As the number of Managers increases beyond these values, the service time decreases and becomes acceptable.

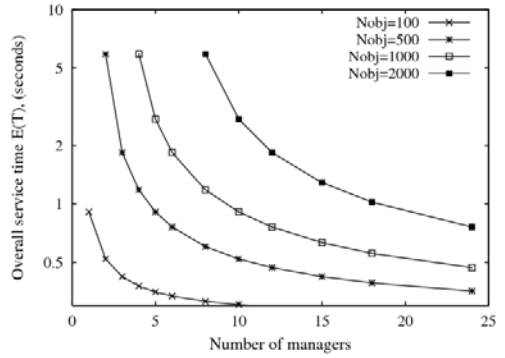

Fig. 4. Overall service time of requests vs. the number of Managers, for different numbers of shared objects. When the corresponding average load is greater than 1 (see Figure 3) the service time is undefined because the system is overloaded and the requests cannot be served.

5.2 Evaluating the Rendezvous load

As described in Section 4, K-link+ relies upon a hybrid paradigm with the simultaneous use of centralized and decentralized communication mechanisms. While the presence of several Managers allows for sharing the processing load pertaining to the management of objects, and brokers are exploited to disseminate objects in a P2P fashion, some high level functionalities are kept centralized. In particular, the maintenance of the Consistency Table and the dynamic assignment of Current Managers to objects is consigned to the Rendezvous.

This choice was made to exploit the efficiency and security of the centralized paradigm at least for such important operations as the two mentioned above. However, the centralized approach can also have two drawbacks: (i) the fault tolerance management and (ii) a possible high load on the server. In order to cope with the first issue, the K-Link+ application manages a possible Rendezvous fault by maintaining a back up Rendezvous that can substitute the current one at each time (this feature is similar to that adopted by JXTA). The second issue is tackled by assigning the Rendezvous only operations that require few computing resources. Indeed, a Rendezvous only copes with metadata documents, which are small and easily manageable, whereas more cumbersome operations, which pertain to the management and update of actual knowledge objects, are distributed among multiple Managers.

To verify the last point, the Rendezvous load was evaluated. It is computed as the contribution of three types of messages (see Table 2): *version update* messages and *manager leave* messages, which are sent by Managers, and *object information request* messages issued by workers when they reconnect. The contribution of *create object* messages is not considered, since it is negligible with respect to others.

The average rates of these three types of messages are computed as described in the following:

1. The average rate of *version update* messages is obtained as follows: (i) the contributions of online and offline requests issued by a single worker, for all their objects (Table 6), are summed; (ii) each time a worker request is accepted by the corresponding Manager, which is assumed to happen 50% of times, a version update message is sent by this Manager to the Rendezvous: therefore the event rate computed at the first step is multiplied by 0.5; (iii) finally, the obtained rate is multiplied by the number of workers.
2. The average rate of *manager leave* messages is obtained by assuming an average connection time of Managers equal to 5 hours. The corresponding rate, equal to 1 message each 18000 seconds, is then multiplied by the number of Managers.
3. The average rate of *object information request* messages is obtained by assuming an average connection time of a worker equal to 3 hours. This rate is then multiplied by the number of workers.

The average time intervals required to process these types of message were estimated on the running K-Link+ application. They are equal to about 50 milliseconds (ms) for processing a *version update* message, and 100 ms for processing a *manager leave* or an *object information request* message. Note that these values are much lower than the processing values experienced by the Manager nodes, since the Rendezvous only deals with metadata information, while the Managers deal with actual knowledge objects. Actually, the load related to the *version update* messages, which depends on the number of shared objects, gives the largest contributions if compared with the load of the other two terms, which do not depend on the number of objects, but on the number of nodes and the connection times of workers and Managers. Figure 5 reports the Rendezvous load and shows that it increases with the number of nodes *N* and the number of shared objects. In this scenario, the CPU utilization of the Rendezvous remains below 65% in all cases. This behavior indicates that to handle up to 100 nodes it is not necessary to adopt a multiple Rendezvous architecture.

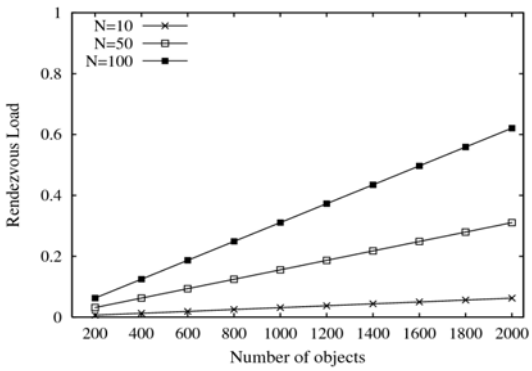

Fig. 5. Load of a Rendezvous node vs. the number of shared objects, with a number of Managers equal to 6.

6. CONCLUSIONS

This paper focuses on two relevant issues in P2P systems, i.e., the consistency of data that arises, since users can work concurrently on multiple replicas of the same object, and the synchronization of peers that is needed when they disconnect from the platform and reconnect again. A model was designed to face these issues in K-link+: a small/medium scale collaborative system founded on the concepts of *Virtual Office* and *workspace*.

A hybrid model was adopted that combines the efficiency of centralized interaction patterns, which are used for the management of metadata information about knowledge objects, with the scalability and adaptive features of decentralized interactions, which are adopted for the update and dissemination of actual data. An analytical performance evaluation, based on the theory of queue networks confirms the suitability of this approach.

In this work, it is assumed that the load is equally shared among the Managers. In a more realistic scenario, each Manager sustains a different load, either because the objects are unfairly distributed, or because different numbers of update requests are issued for different objects. To handle such a situation we are considering a redirection mechanism. In particular, if a Manager experiences a very high load, it can ask the Rendezvous to assign some of the managed objects to a different Manager, in order to alleviate its load. Preliminary experiments are confirming the effectiveness of this approach.

Acknowledgments

This research work is partially funded by the FP6 CoreGRID Network of Excellence which is funded by the European Commission (Contract IST-2002-004265).

7. REFERENCES

[1] Bonifacio, M., Bouquet, P., Mameli, G., Nori, M. KEEx: A Peer-to-Peer Solution for Distributed Knowledge Management. In *Proceedings of the PAKM conference on Practical Aspects of Knowledge Management PAKM '02*, Wien, Austria, December 2002, 490-500.

[2] Chang, T., Ahamad, M.. Improving service performance through object replication in middleware: a peer-to-peer approach. In *Proceedings of the 5th IEEE International Conference on Peer-to-Peer Computing*, Konstanz, Germany, August/September 2005.

[3] Chen, X., Ren, S., Wang, H., Zhang, X. SCOPE: scalable consistency maintenance in structured P2P systems. In *Proceedings of INFOCOM Computer and Communications Societies Conference INFOCOM '05*, Miami, FL, USA, March 2005, 1502-1513.

[4] Chen, Y., Katz, R., Kubiatowicz, J. Dynamic replica placement for scalable content delivery. In Proceedings of the *International Workshop on Peer-to-Peer Systems IPTPS'02*, 2002.

[5] Cohen, E., Shenker, S. Replication strategies in unstructured peer-to-peer networks. In *Proceedings of the ACM SIGCOMM 200202 Conference,* Pittsburgh, PA, USA, August 2002.

[6] Crovella, M. E., and Bestavros. A. Self-similarity in World Wide Web traffic: Evidence and possible causes. IEEE/ACM Transactions on Networking, 5(6), 1997, 835-846.

[7] Crovella, M. E., Taqqu, M. S., and Bestavros. A. Heavy-Tailed Probability Distributions in the World Wide Web. A Practical Guide To Heavy Tails, Birkhauser Boston Inc., Cambridge, MA, USA, 1998, 3-26.

[8] Datta, A., Hauswirth, M., Aberer, K. Updates in highly unreliable, replicated peer-to-peer systems, . In *Proceedings of ICDCS International Conference on Distributed Computing Systems ICDCS '03*, Providence, RI, USA, May 2003, 76-88.

[9] Gopalakrishnan, V., Silaghi, B., Bhattacharjee, B., Keleher, P. Adaptive replication in peer-to-peer systems. In Proceedings *of the 24th International Conference on Distributed Computing Systems ICDCS'04*, Tokyo, Japan, March 2004.

[10] Haverkort, B. *Performance of computer communication systems*. John Wiley & Sons, New York, USA, 1998.

[11] Jain, R. *The art of computer systems performance analysis*. John Wiley & Sons, New York, USA, 1991.

[12] Lamport, L. How to Make a Multiprocessor Computer that Correctly Executes Multiprocess Programs, *IEEE Transactions on Computers*, 28, 9, September 1979, 690-691.

[13] Le Coche, E., Mastroianni, C., Pirrò, G., Ruffolo, M., Talia, D. A Peer-to-Peer Virtual Office for Organizational Knowledge Management. In *Proceedings of the PAKM conference on Practical Aspects of Knowledge Management PAKM '06*, Wien, Austria, November-December 2006, 166-177.

[14] Mastroianni, C., Pirrò, G., Talia, D. Data consistency in a p2p knowledge management platform. In *Proceedings of the second HPDC workshop on Use of P2P, GRID and agents for the development of content networks Upgrade '07*, Monterey, CA, USA, June 2007, 17-24.

[15] Nonaka, I., Takeuchi, H. *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*. Oxford University Press, New York, USA, 1995.

[16] Pathan, A. K., Broberg, J. A., Bubendorfer, K., Kim, H. K., Buyya, R. An architecture for virtual organization (VO)-based effective peering of content delivery networks. In *Proceedings of the second HPDC workshop on Use of P2P, GRID and agents for the development of content networks Upgrade '07*, Monterey, CA, USA, June 2007, 29-38.

[17] Ramamritham, K., Prashant, S. Dynamic Information Dissemination, *IEEE Internet Computing*, 11(4), July/August 2007, 14-15.

[18] Rosenkrantz, D. J., Stearns, R. E., Lewis, P. M. System level concurrency control for distributed database systems, A*CM Transactions on Database Systems*, 3(2), 1978.

[19] Tanenbaum, A. S., van Steen, M. *Distributed Systems: Principles and Paradigms*, Prentice Hall, 2002.

[20] Traversat, B., Abdelaziz, M., Pouyoul E. Project JXTA: A Loosely-Consistent DHT Rendezvous Walker. Available at http://www.jxta.org/docs/jxta-dht.pdf.

[21] Wang, Z., Kumar, M., Das, S. K., Shen, H. File Consistency Maintenance Through Virtual Servers in P2P Systems. In *Proceedings of the IEEE Symposium on Computers and Communications ISCC '06*, Cagliari, Italy, June 2006, 435-441.

[22] Wang, C., Xiao, L., Liu, Y., Zheng, P. DiCAS: An Efficient Distributed Caching Mechanism for P2P Systems, *IEEE Transactions on Parallel and Distributed Systems*, 17(10), 2006.