Adaptive Resource Management in Peer-to-Peer Middleware

Thomas Repantis, Yannis Drougas, Vana Kalogeraki*
Department of Computer Science & Engineering
University of California, Riverside
Riverside, CA 92521
{trep, drougas, vana}@cs.ucr.edu

Abstract

Supporting distributed real-time applications in large-scale and heterogeneous distributed environments has attracted much attention recently. However, the decentralized and dynamic characteristics of the systems present a number of challenges in managing the processor and networking resources and scheduling the application execution across multiple peers. In this paper, we address these problems with a novel resource management architecture for scheduling soft real-time applications in overlay networks. Key to our approach is that nodes maintain a partial view of resource availability of their peers in local profiles, keep track of the urgency of the requests and balance application execution across multiple nodes. Our techniques are entirely decentralized and use only local knowledge considering unpredictable latencies and changing resource availability.

1. Introduction

Distributed applications (such as multimedia, telecommunications, business enterprises and tele-medicine) are increasingly being deployed in large-scale and heterogeneous environments. These applications demand multiple end-to-end quality of service (QoS) guarantees, such as predictable latency and jitter, reliability, and scalability. Hosting these applications on wide-area environments with unpredictable latencies and changing resource availability, requires systems that are easy to manage and able to adapt to dynamic changes in the utilization or availability of the resources.

Distributed object middleware has received a lot of attention in the past for providing a common programming abstraction that enables applications to interoperate independently of their programming languages, computing platforms and networking protocols. Middleware has evolved in the last few years to provide a mature, extensive and portable infrastructure, standard protocols and key fundamental services.

Although current middleware provide *necessary* mechanisms to meet the end-to-end QoS requirements of the applications, they are not *sufficient* by themselves to manage large-scale distributed real-time systems [23]. This limitation is due to (a) *inadequacy* of a central manager to obtain an accurate global view of the applications and the current resource usage in the system, (b) the *distributed nature* of the applications that execute across multiple processors, (c) *unpredictability* in the arrival times of the application execution, (d) *sharing* of heterogeneous computer resources by many applications, and (e) *multiple QoS requirements* that need to be satisfied simultaneously and traded-off.

In our view, the new emerging Peer-to-Peer (P2P) model [14] has inherent advantages including scalability, decentralization and ease of use. The P2P paradigm offers the opportunity to harness the computational power available at the edges of the Internet for providing and using distributed services, in a large scale and in a decentralized fashion. The P2P model represents the natural evolution of the client-server model that was primarily used in small-scale distributed environments, to accommodate large numbers of applications and users. The most distinct characteristic of the P2P model is that there is symmetric communication between the peers; each peer has both a client and a server role. P2P systems achieve scalability by enabling direct and real-time communication among a large number of applications and resources located in geographically distributed areas. They provide decentralization by enabling the making of local decisions, without the need for a global manager. These characteristics however make them extremely difficult to manage, and therefore it is difficult to provide QoS guarantees to the applications.

An example of a distributed application is media streaming and transcoding. This application requires the transfer of multimedia data from the sources to the receiver. The audio and video streams may need to be customized such

* This research has been supported by NSF Award 0330481.

as transcoded to different formats or presentations (e.g., lower resolution) to bring the data to different devices or to meet the particular QoS requirements of the individual users. Transcoding allows for adaptation to specific spatial and chroma resolution requirements of the receiver, such as those that may be imposed by energy-constrained mobile devices. These pose end-to-end soft real-time and QoS requirements on data transmission, including fast and reliable transfer, and substantial throughput. To support the QoS demands of the distributed applications, we need a resource management system that is adaptive, flexible and scalable.

In this paper we propose an adaptive resource management system for large-scale distributed real-time systems. We propose two mechanisms to meet the end-to-end soft real-time and QoS requirements of the distributed applications. These mechanisms are decentralized, adaptive and use only local information. First, we propose service graphs that allow the making of local decisions by capturing application behavior, object dependencies, communication latencies and resource overheads, therefore significantly improving scalability. Second, we propose dynamic and adaptive load balancing algorithms that exploit underutilized resources and examine the execution times and resource requirements of the applications to meet their end-to-end soft real-time and QoS requirements, and therefore significantly improving their performance.

The fundamental difficulty in large-scale distributed real-time systems comes from the decentralized nature of the systems, where no peer can have a global view of the system making it difficult to schedule and load balance distributed applications. Furthermore, the dynamic character of the system, where nodes may connect, disconnect or fail unexpectedly imposes challenges in providing real-time guarantees. The problem is complicated further by the heterogeneity of the peers, in terms of processing power, network connectivity, and available software. In such an environment, a static application allocation –no matter how clever it might be– will not suffice. More specifically, this paper makes the following contributions:

- We propose a decentralized resource management architecture for a large-scale real-time distributed system. The system builds application service graphs and a resource graph to meet application QoS requirements.

- We propose a load balancing algorithm based on the notion of fairness. The algorithm ensures that the load among the peers is fairly balanced. Our proposed schemes scale well with respect to the number of peers and work effectively in a heterogeneous and dynamic environment.

2. System Architecture

We model a large-scale distributed system as an overlay network of peers grouped into domains according to their topological proximity, and Resource Managers are selected among regular peers to act as leaders of the domain. Hence, a domain is structured as a single Resource Manager for the domain and Connection Managers, Profilers and Schedulers for each of the processors in the domain. Each Resource Manager is connected to all the processors of its domain directly. The Resource Manager has a global view of the domain in terms of the applications in the domain and the utilization of the system resources. The responsibility of the Resource Manager is to distribute the application objects on the processors to meet the application QoS requirements. The Resource Manager is also responsible for collecting load information of all the peers of the domain, used to allocate objects to the less loaded peers in the domain.

The Connection Manager is responsible for managing the peer connections; that is, establishing or destroying connections of the processor to other peers. The number of connections is typically limited by the resources at the peer. The Profiler on the processor is responsible for measuring the current processor and network load of the peer and monitoring the computation and communication times of the applications as they execute. The Profiler measurements will be propagated to the Resource Manager of the domain to construct the application service graphs and the resource graph of the domain. The Local Scheduler of every peer determines the execution sequence of the applications at the peer based on the scheduling algorithm implemented in the system. Our scheduling algorithm is based on the Least Laxity Scheduling (LLS) algorithm [4] that exploits the deadlines of the applications and the actual computation and execution times on the processors to determine an efficient schedule for the applications in the system. This allows us to capture timing delays as the applications execute across multiple processors and adjust their execution order.

3. Information Base

Our system maintains current information at three different levels. On a *domain-level*, Resource Managers maintain current resource utilization about all the nodes in their domain and end-to-end application information, collected at run-time. On a *system-level*, Resource Managers maintain summarized information about the other domains, updated lazily using a gossiping protocol. On a *peer-level*, nodes maintain local resource loads and local application execution and communication times.

3.1. Resource Manager Data Structures

Every Resource Manager maintains the following information for its domain:

1. A unique identifier RM for the Resource Manager of the domain.
2. The identifiers of the processors P_i in the domain; each processor is characterized by a unique ID (such as the pair $< IP_i, port_i >$ or a randomly generated number).
3. The current processor load l_i for all processors i, expressed as the product of processing power with current utilization.
4. The currently used network bandwidth bw_i for all processors i.
5. The application objects O_{ij} located on processor i. For example, for a transcoding application, these would be media objects and their characteristics are also stored as meta-data (hash value, bitrate, resolution, codec).
6. The services S_{ij} each processor can offer. For example, for a transcoding application, these would be the transcoding services available in each processor.
7. The application service graphs that represents the applications currently executing and the resource graph that shows the domain's resources and their usage.

In addition, each Resource Manager maintains summarized information for other domains, propagated to it by its Resource Manager peers. More specifically, it stores:

1. A list of other domains D_k in the system, along with the identifiers of the corresponding Resource Managers RM_k.
2. A summary of the available application objects $SumO_k$ and the available services $SumS_k$ in each domain. The summaries can be obtained using Bloom Filters [19].

3.2. Peer Data Structures

The Profiler on each processor i maintains the following information:

1. The data objects O_i allocated locally, e.g. the media files it has stored.
2. The services S_i it can offer, e.g. the transcoders available in this processor.
3. The current processor load l_i.
4. The currently used network bandwidth bw_i.
5. The current dependencies between this processor and other peers, meaning which peers are currently receiving services by this peer or offering services to this peer.

3.3. Application Service Graph

A distributed application consists of a number of application tasks. We model an application task as a sequence of invocations of objects and services distributed across multiple processors. The execution of the application is triggered by users and multiple tasks invoked by different users can be executed concurrently in the system.

We use an application service graph G_s to represent the objects and services invoked by the application tasks, and the corresponding resource requirements, as shown in Figure 1(B). In general, a distributed application might require a sequence of service invocations across multiple peers before the result can be delivered to the peer receiving the service. The vertices of the service graph represent objects or services of the system, while the edges represent connections between the peers, which have been established for receiving the services.

The Resource Manager keeps the following information for each application task t:

- $Deadline_t$: the time interval, starting at task initiation within which the task should complete, specified by the end user.
- $Importance_t$: a metric that represents the relative importance of the application, specified by the end user.
- $Execution\_time_t$: the estimated amount of time from initiation to completion of application task t. The execution time is computed as the sum of the processing times of the objects and services on the processors and their communication times as they are invoked by the tasks. Note that the execution time of a distributed application also depends on the number of applications in the system and the load on the resources. Our goal is to maximize the number of applications that meet their deadlines.

3.4. Resource Graph

The Resource Manager also maintains a resource graph G_r for its domain D, as shown in Figure 1(A). G_r is a directed graph. Each vertex v of G_r represents an application state, while each edge e represents a service, accompanied by its current load.

The aim of the resource graph G_r is to identify the services available at the peers and assign those services to the requesting applications. The resulting assignment should meet the application deadline and fulfill its resource requirements. Let E_{in} be the set of G_r's edges ending to v and E_{out} be the set of G_r's edges beginning from v. The application has the option to follow any of the edges $e \in E_{out}$ beginning from v, leading to the invocation of a different service.

To decide which service to invoke, enough CPU and bandwidth must be available for the execution of the service. Thus, given an initial state x_{init}, we can determine the different allocation states we can end up to, by traversing G_r. Using the resource graph allows us to formulate the problem as a search problem and directly apply well-studied algorithms [22] in order to solve our problem.

4. System Operation

4.1. Overlay Construction

The peers are divided in geographical domains. The only parameter determining the domain size is the maximum number of processing peers a Resource Manager can manage, depending on its own capabilities. As described in this section, once a Resource Manager reaches the maximum number of peers it can manage, a new domain is created. Thus, the network is unstructured and self-organizing.

A peer must demonstrate that it has sufficient resources and stability before it can qualify for becoming a Resource Manager for its domain. If a peer meets the qualifications, it *may* become a Resource Manager, depending on the network topology and the needs in the domain it belongs to. The requirements for becoming a Resource Manager are: i) Sufficient bandwidth, ii) Sufficient processing power, iii) Sufficient uptime. According to how affluent a peer is in those resources, it is assigned a score, that determines its position in the list of peers in the domain that are eligible for becoming Resource Managers. The first peer in the list serves as backup Resource Manager, keeping an up-to-date copy of all the information the Resource Manager stores. This is achieved by receiving periodic updates from the primary Resource Manager.

The protocol used for connecting to the network is analogous to the ultrapeer negotiation utilized in Gnutella 0.6 [8]. When a new peer joins the network, it connects to the Resource Manager of its geographical domain, or to a random peer who redirects it to the Resource Manager. If the Resource Manager has available bandwidth and processing power, it accepts the processor in its domain, and adds it to the list of potential Resource Managers, if it qualifies. If the Resource Manager has reached the maximum number of processors it can support, it accepts the newcomer as a new Resource Manager if it qualifies, otherwise it redirects it to a Resource Manager of another domain. Peers may disconnect from the system either intentionally or due to a failure. When a processor disconnects, the Resource Manager of its domain will sense the withdrawn connection and update the available data objects and services in the system to include the change. The resource graph is also updated, by removing the edges that were referring to the services offered by the particular peer. If the service graph included the peer in question as one of its vertices, this means that an application task has been interrupted. The Resource Manager must then not only remove the vertex from the service graph, but also find a peer to substitute it. This is done following the task allocation algorithm described later.

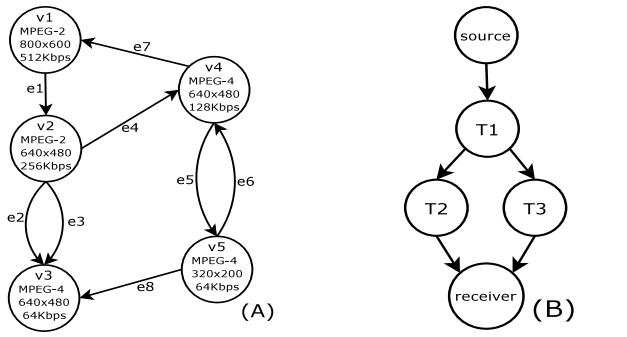

Figure 1. Example of a Resource Graph (A) and of one Service Graph (B) produced by it.

When a Resource Manager disconnects, the backup Resource Manager senses the withdrawn connection. It then takes over as a Resource Manager, using its backup copy of the Resource Manager information. It also selects the next available processor from the list of processors that qualify for becoming Resource Managers and uses it as a backup Resource Manager, providing it with copies of the resource management information.

4.2. Load Balancing Strategy

Each Resource Manager is responsible for composing the application service graph G_s, so that the QoS requirements of the application will be met. In addition, the Resource Manager needs to make sure that the load among the peers of its domain is fairly balanced. In order to evaluate how fair a particular load distribution is, the Resource Manager employs the *Fairness Index* metric [9].

The Fairness Index of the load distribution $\bar{l}$ among the peers of a domain D is defined as:

$$\mathcal{F}(\bar{l}_{P_D}) = \frac{(\sum_{p \in P_D} l_p)^2}{|P_D| \cdot \sum_{p \in P_D} l_p^2} \quad (1)$$

where P_D is the set of peers in D and l_p represents the load of peer p. Essentially, the fairness index is an absolute index that can quantify the uniformity of the load distribution $\bar{l}_{P_D}$ across P_D. The Fairness Index value ranges between values 0 and 1. Given the above equation, one can verify that the higher the value of the Fairness Index, the more uniform (fair) the distribution is. A totally fair sys-

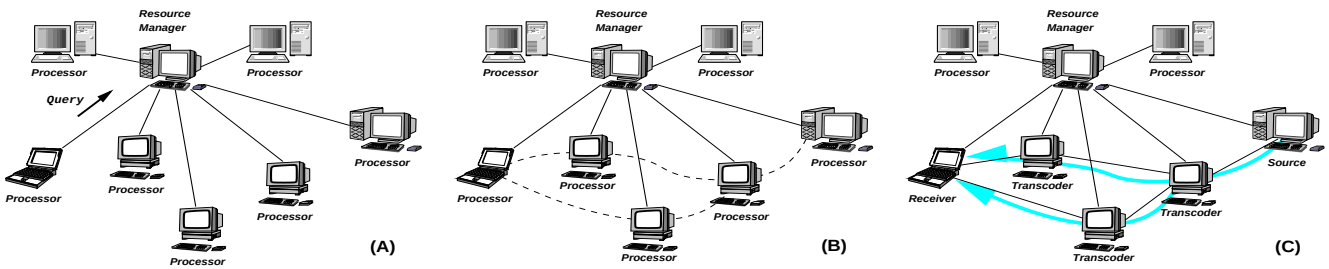

Figure 2. Example of application task assignment: (A) A peer submits a query to the Resource Manager. (B) The Resource Manager assigns the task to peers. (C) Transcoded media streaming begins.

tem has a Fairness Index of 1, while a totally unfair system has an index value of 0. A value of 0.1 indicates the system to be fair to only 10% of the users and unfair to 90% of the users. Thus, the Resource Manager can easily compare the fairness of two different states of D by just comparing the Fairness Index of $\bar{l}_{P_D}$.

Other useful properties of the Fairness Index are that it does not depend on the size, nor the scale of the load distribution. For our system, this means that the Resource Manager can have a clear picture of how fair the load distribution is across the peers. For example, when the fairness is low, the Resource Manager could take some action, such as use the same service from another peer.

An additional feature of the Fairness Index is that it does not necessarily monotonically increase of decrease when the load l_p of a single peer p increases or decreases. Assuming that the load on the rest of the peers in P_D remains the same, $\bar{l}_{P_D}$ increases when the load approaches a specific value l_{best} (where $\bar{l}_{P_D}$ is maximized) and decreased when the load diverges from l_{best}. Intuitively, this tells us that there is no fair (with high Fairness Index) load distribution where some of the peers are overloaded or underloaded compared to the rest of the peers in D.

The Fairness Index is a continuous quantity, modified when the loads of some of the peers change. No matter how small a change of a load is, it is going to affect the Fairness Index of the load distribution. For example, the fairness of a distribution decreases when the load of a single element diverges from the value l_{best} that results in the maximum Fairness Index (assuming that the rest of the loads do not change), while it increases in the opposite case.

The advantage of the Fairness Index metric is that it captures the load of a peer compared to the load of other peers in the domain. If the load of all the peers in the domain is high, this indicates an overloaded system where the addition of another application will increase the execution times of the existing applications and may cause them to miss their deadlines. In these cases, the Resource Manager will forward the application request to other Resource Manager peers.

Finding the optimal distribution of the load to the peers is known to be NP-complete. However, recent work shows that, in a centralized context (that is, when a peer knows all the individual loads and the resource availability of all the peers), a greedy algorithm can give good results [17].

4.3. Task Allocation Algorithm

An example of the operation of our system is shown in Figure 2. The system provides on-demand application task execution, where a user at a peer submits a query to the resource manager of its domain, for a task t, specifying its name (id_t) and QoS requirements (such as $deadline_t$). In order for the application task to be completed several objects and services may need to be employed. For example, in a transcoding application a peer might ask for a media object by name, also specifying a set of acceptable bitrates, resolutions and codecs. Several transcoding services might need to be employed before the media can be delivered in one of the requested formats.

The Resource Manager will then try to allocate resources for t in such a way that these requirements are met and the fairness of the load distribution is maximized. The system will only try to allocate services for the new task, avoiding to move services that accommodate already running tasks.

The Resource Manager uses the resource graph to search for configurations in the domain that satisfy the requested service. The Resource Manager uses the Breadth-First-Search (BFS) algorithm to search for services (edges) connecting the initial and final requested application states (vertices), v_{init} and v_{sol}. It prunes the possible solutions using the requested QoS requirements q. It calculates which paths satisfy the deadline by utilizing the current load information. Among the allocations that sat-

algorithm ALLOCATIONALGORITHM
input: a task T, a requirement set q
output: a task execution sequence Seq, a load fairness f
Let v_{init} be the vertex of G_r representing the original state
Let v_{sol} be the vertex of G_r representing the required output state
Let Seq, e_{seq} be two empty task execution sequences
Set $f_{max} \leftarrow 0$
Let $queue$ be a queue containing a single element v_{init}
Let $queue_{seq}$ be a queue containing a single
empty task execution sequence
while $queue$ is not empty **do**
Set v equal to the first element of $queue$
Set e_{seq} equal to the first execution task sequence of $queue_{seq}$
if v has not been visited before **and**
execution task sequence e_{seq} fulfills requirements in q **then**
if $v = v_{sol}$ **then**
Let f be the load fairness of the peer load distribution
after the task sequence e_{seq} is assigned to the peers
if $f > f_{max}$ **then**
Set $Seq \leftarrow e_{seq}$
Set $f_{max} \leftarrow f$
end if
else for each edge e from v to another vertex v' of G_r **do**
Add v' to the end of $queue$
Add e to the end of e_{seq}
Add e_{seq} to the end of $queue_{seq}$
end if
end if
Remove the first elements of $queue$ and $queue_{seq}$
end while
return $\{Seq, f_{max}\}$
end algorithm

Figure 3. The task allocation algorithm.

isfy the QoS requirements, the algorithm returns the one that results to the maximum fairness of the load distribution among the peers. If no allocation that satisfies the given QoS exists, the algorithm reports that. The complete algorithm is shown in Figure 3.

After the Resource Manager has solved the search problem described, and determined the service graph for that particular session, it initiates the actual task allocation. In particular, graph composition messages are sent to the nodes that will participate in the streaming graph, allowing them to establish the appropriate connections. Then, the distributed application can begin.

In Figure 1, an example resource graph and a service graph for a transcoding application are shown. Let us assume a source that is transmitting 800x600 MPEG-2 video, at 512 Kbps and a user that wants to view that video in 640x480 MPEG-4, at 64Kbps. Our goal is to find a path from v_1 (which represents the format of the source) to v_3. In this example, we can follow any of the $\{e_1, e_2\}$, $\{e_1, e_3\}$ or $\{e_1, e_4, e_5, e_8\}$. Our load balancing algorithm will compute which of these paths satisfy the QoS requirements of the application and result in a fair resource allocation. Assuming that the sequences of the edges $\{e_1, e_2\}$ or $\{e_1, e_3\}$ meet the QoS requirements and result in a fairly loaded execution, the Resource Manager will construct the service graph of Figure 1(B). In the service graph G_s, vertices T_1, T_2 and T_3 are the transcoders that are represented by edges e_1, e_2 and e_3 in G_r respectively. Also, while G_r represents the number of available services and current resource usage in the system, every produced G_s refers only to a particular application task execution.

4.4. Resource Utilization Feedback

The Resource Manager of every domain in addition to maintaining lists of the available objects and services in its domain, also collects resource utilization feedback from the processors. For other domains, just summaries of the available objects and services are kept.

Intra-Domain Propagation. The Profiler of every peer in a domain monitors current processor and network load, as well as computation and communication times of the executing applications. This information is periodically propagated to the Resource Manager of the domain. Care must be taken when selecting the period for the load updates propagation. Too frequent updates would cause high network traffic and processing load, while too infrequent updates may not capture the application requirements adequately. The application QoS requirements determine the appropriate update frequency.

Inter-Domain Propagation. In addition to information about its own domain, each Resource Manager maintains summarized information of the available objects and services available in other domains. The summaries of those have to be updated only when peers join or leave the system. Hence, a gossiping protocol (similar for example to the one used in [29]) should suffice for lazily propagating changes among the Resource Managers.

4.5. Adaptive Resource Management

During a service session a variety of global and local factors may affect the performance of the system:

- *Changes in the infrastructure*. The availability of resources is affected by infrastructure changes. As transient nodes fail or disconnect from the system, the execution of one or more application tasks may be affected. Hence, the service graph may need to be reconstructed in order to recover from such failures.
- *Changes in the number of service sessions*. As the number of concurrent service sessions increase or decrease over time, processors need to make adaptive

decisions to balance their load. Moreover, some local load balancing decisions might increase the execution times of other existing application tasks.

- *Changes in the local resource conditions*. In case of local resource overload, the performance of an application task's execution degrades as well. Besides the dynamic changes in the user requirements, overload conditions could also be caused by extraneous workload or network traffic.
- *Changes in the user requirements*. Users may change QoS requirements dynamically. Specifically, they may reduce the requested bit-rate or relax their deadlines to cope with congested networks, or increase the QoS parameters if they assume resources are abundant.

All the aforementioned factors not only make it hard to find an optimal initial application task allocation, but can also result in degrading performance during the application task execution. This calls for clever initial application task allocation together with adaptive application task reassignment. Both are carried out by the domain Resource Managers.

When admitting a new application task the resource manager estimates whether its QoS requirements can be accommodated by the system's current resources without overloading the system. If all peers are too loaded to provide the requested QoS guarantees, then the task is not admitted, since it would not reach completion on-time and would also harm the performance of the currently executing tasks. Instead, the task query is redirected to a Resource Manager of another domain. To maximize the probability that the task will be admitted, the summaries of the available objects and services in other domains are utilized to direct the query to the appropriate domain.

When the Resource Manager determines that the system is overloaded (for example if the processor or network load is constantly above a certain threshold for all peers or if the applications do not meet their deadlines), some of the currently running application tasks might be reassigned. The allocation algorithm described in section 4.3 is run again, or tasks are redirected to other Resource Managers if all peers of the domain are overloaded. The new connections can be established offline, to hide the overhead.

5. Related Work

Traditional peer-to-peer systems, both structured (Distributed Hash Tables) [27, 18, 21] and unstructured [8], have focused on data sharing applications. Constructing overlay networks that enable peers to provide and use distributed services poses however several challenges that do not arise in file sharing applications. In [4] we have focused on the scheduling of tasks in such systems, while in [5] we have described a completely decentralized media streaming and transcoding architecture, utilizing local quality adaptation and feedback-based coordination. The problem of deploying and managing federated systems has also been discussed in [6], where agents gather information about the system state and services and exchange it periodically using a gossiping protocol.

Work has been done to enhance ORBs with load balancing capabilities, supporting a variety of adaptive and non-adaptive strategies [3], [16]. In addition, several efforts have focused on extending middleware architectures with real-time and QoS capabilities [31], [30], [24]. These have mainly focused on mechanisms for specifying QoS parameters and enforcement of timing constraints for hard real-time systems using rate monotonic scheduling. Gill *et al* [7] proposed a dynamic scheduling strategy which provides scheduling assurance for critical application tasks while offering the flexibility to optimize the use of scarce resources.

In [12], [11], [2] multiple types of resources are allocated to the applications. Jensen *et al* [10] propose soft real-time scheduling algorithms based on application benefit, with the goal to maximize the overall system benefit. Stankovic *et al* [26] use value-based function based on importance and timing requirements to schedule application tasks.

Many researchers have realized the need for systems that can adapt to dynamic, unpredictable changes in the computing environment [13], [20], [25], [28], [1], [15], [29].All of them agree that (i) adaptive systems need integrated monitoring, dynamic execution time prediction and scheduling across multiple processors, and (ii) efficient dynamic scheduling algorithms require knowledge of real-time application task information including the application task's deadline, reliability and resource requirements.

6. Conclusion

We have described an adaptive resource management architecture for large-scale, real-time, peer-to-peer middleware. Our system utilizes resource managers that maintain service graphs representing the currently executing tasks on the system and a resource graph representing the available resources and their current usage. We proposed a load balancing algorithm to achieve a fair utilization of the system resources. Our proposed architecture scales well with respect to the number of peers and works effectively in heterogeneous and dynamic environments.

References

[1] T. Abdelzaher, J. Stankovic, C. Lu, R. Zhang, and Y. Lu. Controlware: A middleware architecture for feedback control of software. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, July 2002.

[2] C. Aurrecoechea, A. T. Campbell, and L. Hauw. A survey of QoS architectures. *Multimedia Systems*, 6(3):138–151, 1998.

[3] J. Balasubramanian, D. Schmidt, L. Dowdy, and O. Othman. Evaluating the performance of middleware load balancing strategies. In *Proceedings of the 8th International IEEE Enterprise Distributed Object Computing Conference*, 2004.

[4] F. Chen and V. Kalogeraki. RUBEN: A technique for scheduling multimedia applications in overlay networks. In *Proceedings of the IEEE Global Telecommunications Conference 2004 (Globecom'04)*, November 2004.

[5] F. Chen, T.Repantis, and V. Kalogeraki. Coordinated media streaming and transcoding in peer-to-peer systems. In *Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS 2005)*, April 2005.

[6] F. Cuenca-Acuna and T. Nguyen. Self-managing federated services. In *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems (SRDS 2004)*, 2004.

[7] C. Gill, D. L. Levine, D. C. Schmidt, and F. Kuhns. The design and performance of a real-time CORBA scheduling se rvice. *International Journal of Time-Critical Computing Systems*, 23(2):221–264, June 1991.

[8] Gnutella Protocol Development . http://rfc-gnutella.sourceforge.net/, 2003.

[9] R. K. Jain, D.-M. W. Chiu, and W. R. Have. A quantitive measure of fairness and discrimination for resource allocation in shared computer systems. Technical Report DEC-TR-301, Digital Institution Corporation, Hudson, MA 01749, September 26 1984.

[10] E. D. Jensen, C. D. Locke, and H. Tokuda. A time-driven scheduling model for real-time operating systems. In *Proceedings of the IEEE Sixth Real-Time Systems Symposium*, pages 112–122, San Diego, CA, December 1985.

[11] T. F. Lawrence. Quality of service (QoS) a model for information. In *Proceedings of the Fourth International Workshop on Object-Oriented Real-Time Dependable Systems*, pages 180–182, Santa Barbara, CA, January 1999.

[12] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, and J. A. Hansen. Scalable solution to the multi-resource QoS problem. In *Proceedings IEEE 20th Real-Time Systems Symposium*, pages 315–326, Phoenix, AZ, December 1999.

[13] G. Manimaran and C. R. R. Murthy. An effi cient dynamic scheduling algorithm for multiprocessor real-time systems. *IEEE Transactions on Parallel and Distributed Systems*, 9(3):312–319, March 1998.

[14] D. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-Peer Computing. Technical report, HP Technical Report, HPL-2002-57, 2003.

[15] K. Nahrstedt and R. Steinmetz. Resource management in networked multimedia systems. *Computer*, 28(5):52–63, May 1995.

[16] O. Othman and D. Schmidt. Optimizing distributed system performance via adaptive middleware load balancing. In *Proceedings of the ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems (OM 2001)*, 2001.

[17] P. Raftopoulou. Fair Resource Allocation in P2P systems: Theoretical and Experimental Results. Master's thesis, Department of Electronic and Computer Engineering, Technical University of Crete, Greece, 2003.

[18] S. Ratnasamy, P. Francis, M. Handley, and R. Karp. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM 2001*, August 2001.

[19] T. Repantis and V. Kalogeraki. Data dissemination in mobile peer-to-peer networks. In *Proceedings of the 6th International Conference on Mobile Data Management (MDM 2005)*, May 2005.

[20] D. I. Rosu, K. Schwan, and R. Jha. On adaptive resource allocation for complex real-time applications. In *Proceedings of the IEEE 18th Real-Time Systems Symposium*, pages 320–329, San Francisco, CA, 12 1997.

[21] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms*, November 2001.

[22] S. Russell and P. Norvig. *Artifi cial Intelligence: A Modern Approach*. Prentice-Hall, 2nd edition, 2003.

[23] R. Schantz, J. P. Loyall, C. Rodrigues, D. Schmidt, Y. Krishnamurthy, and I. Pyarali. Flexible and adaptive QoS control for distributed real-time and embedded middleware. In *ACM/IFIP/USENIX International Middleware Conference*, Rio de Janeiro, Brazil, June 2003.

[24] D. C. Schmidt, D. Levine, and S. Mungee. The design of the TAO real-time object request broker. *Computer Communications*, 21(4):294–324, April 1998.

[25] J. Schonwalder, S. Garg, Y. Huang, A. van Moorsel, and S. Yajnik. A management interface for distributed fault tolerant CORBA services. In *Proceedings of the IEEE Third International Workshop on Systems Management*, pages 98–107, Newport, RI, April 1998.

[26] J. A. Stankovic and K. Ramamritham. The Spring kernel: A new paradigm for real-time operating systems. *Operating Systems Review*, 23(3):54–71, July 1989.

[27] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM 2001*, August 2001.

[28] J. J. Sydir, S. Chatterjee, and B. Sabata. Providing end-to-end QoS assurances in a CORBA-based system. In *Proceedings of the IEEE 1st International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 53–61, Kyoto, Japan, 4 1998.

[29] R. van Reness, K. Birman, and W. Vogels. Astrolabe: A robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining. *ACM Transactions on Computer Systems*, 21(2):164–206, May 2003.

[30] V. F. Wolfe, L. C. DiPippo, R. Ginis, M. Squadrito, S. . Wohlever, I. Zykh, and R. Johnston. Real-time CORBA. In *Proceedings of the IEEE 3rd Real-Time Technology and Applicati ons Symposium*, pages 148–157, Montreal, Quebec, Canada, June 1997.

[31] J. Zinky, D. Bakken, and R. Schantz. Architectural support for quality of service for CORBA objects. *Theory and Practice of Object Systems*, 3(1):55–73, 1997.