Load Balancing Techniques for Distributed Stream Processing Applications in Overlay Environments

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

Abstract

Service overlays that support distributed stream processing applications are increasingly being deployed in wide-area environments. The inherent heterogeneous, dynamic and large-scale nature of these systems makes it difficult to meet the Quality of Service (QoS) requirements of the distributed stream processing applications. In this paper we address the load balancing problem for distributed stream processing applications and present a decentralized and adaptive algorithm that allows the composition of distributed stream processing applications on the fly across a large-scale system, while satisfying their QoS demands. The algorithm fairly distributes the load on the resources and adapts dynamically to changes in the resource utilization or the QoS requirements of the applications. Our experimental results demonstrate the scalability, efficiency and performance of our approach.

1 Introduction

Advances in processing and communication technologies have resulted in the development of *large-scale service overlays* (or *Peer-to-Peer* overlays). The new emerging Peer-to-Peer (P2P) model has become a very powerful paradigm for developing Internet-scale systems and sharing resources (i.e., CPU cycles, memory, storage space, network bandwidth) over large scale geographical areas. Peer-to-Peer overlays are logical networks of many nodes (peers), constructed on top of heterogeneous operating systems and networks. Such overlays are flexible and deployable approaches that allow users to perform distributed operations without modifying the underlying physical network. These have found popular applications in a number of domains including file sharing, content distribution, multimedia streaming, multicast and distributed games.

In the last few years, a new class of *distributed stream processing applications* have emerged in domains such as network traffic monitoring, financial, healthcare, sensor data acquisition and multimedia [3, 1, 9, 14]. In distributed stream processing applications, data produced by heterogeneous, autonomous and large numbers of globally-distributed data sources are composed dynamically to generate results of interest. These offer scalability and availability advantages by harnessing distributed processing elements in a cost-effective way. More advantages of distributed stream processing applications include their ability for customized delivery, for adaptation to different loads, and for resiliency to node failures. Distributed stream processing can also be applied to multimedia streams, to eliminate the need for a dedicated server with a high bandwidth connection and offer media services that can be composed on demand [11].

Several characteristics make the provisioning of real-time and QoS support to distributed stream processing applications on large-scale service overlays a challenging problem. First, overlay nodes are typically heterogenous in terms of processor capacity, network inbound/outbound bandwidth, and software. Second, applications have multiple QoS demands including high throughput, small delay and jitter. Third, applications are composed at run-time, without a priori notification, posing stringent resource requirements on processor cycles and available network bandwidth along the streaming paths. As a result, the quality of the services may vary with time in an unpredictable way.

Current research does not focus on deciding the locations of the services when composing a distributed

stream processing application dynamically to satisfy the end-to-end application QoS demands. This is a difficult problem in large-scale distributed environments, where well-established centralized solutions [15] cannot be applied directly. However, resource allocation becomes an important factor that affects the scalability of service overlays and the performance of the applications when deployed on a shared and heterogeneous infrastructure.

In this paper we address the load balancing problem in large-scale service overlays. We propose an adaptive and scalable load balancing technique for fair allocation of resources in large-scale service overlays, so that the QoS demands of distributed stream processing applications are satisfied. We present multimedia streaming and transcoding as a service example, but our techniques apply to any large-scale distributed application. Our approach allows composition of distributed stream processing applications dynamically, to satisfy their end-to-end QoS demands with high probability.

Our Contributions: (1) We demonstrate a decentralized and fair resource allocation algorithm that distributes the processing and communication load fairly among the nodes of a large-scale system while meeting application end-to-end QoS demands. To react to dynamic changes in the resource utilization or the application behavior, quality adaptation mechanisms are used to trade off quality level with resource usage. (2) We present experimental results, using a distributed media streaming and transcoding application over a large-scale overlay, to demonstrate the performance, efficiency and scalability of our techniques.

2 System Model

2.1 Peer-to-Peer Network Model

Service overlay networks are logical networks of nodes (peers) constructed on top of the physical network, in which the nodes are linked through virtual connections (figure 1). Each peer p is identified by the IP address of the physical node it resides and the port it is listening to. Each node keeps a small number of connections to other peers; the number of connections is typically limited by the network bandwidth at the peer. The peers of a node can be randomly selected, defined a priori based on some optimization criteria (such as round-trip delays), or dynamically established and revised in response to the node interactions or changes in the processing and networking conditions [8].

Each peer p offers the set of services S_p and is constrained by the CPU speed $cycles_{total,p}$ and the limited amount of bandwidth $bps_{total,p}$ of the node. The communication link between any two nodes p and q is $band_{pq}$, where $\sum_q band_{pq} \leq bps_{total,p}$. This leads to a limit on the number of connections a node can maintain to other peers. We denote the number of connections as $conn_p$. To compute the load of peer p, we consider both its processing load and network bandwidth. Thus, we compute the load of peer p as the weighted sum of its current processing and communication load, as follows: $l_p = w_c \cdot cpu_p + w_b \cdot band_p$, where $cpu_p = \frac{cycles_{used,p}}{cycles_{total,p}}$ and $band_p = \frac{bps_{used,p}}{bps_{total,p}}$ are the portions of the processing power and bandwidth currently being used at p. To give higher weight to the scarcest of the two resources (processing power or bandwidth), we define the weights as follows: $w_c = min(1, max(0, 0.5 + cpu_p - band_p))$ and $w_b = min(1, max(0, 0.5 + band_p - cpu_p))$. The weights w_c and w_b take values between 0 and 1 and have a sum of 1; thus, the weighted load $load_p$ of every peer p is between 0 and 1.

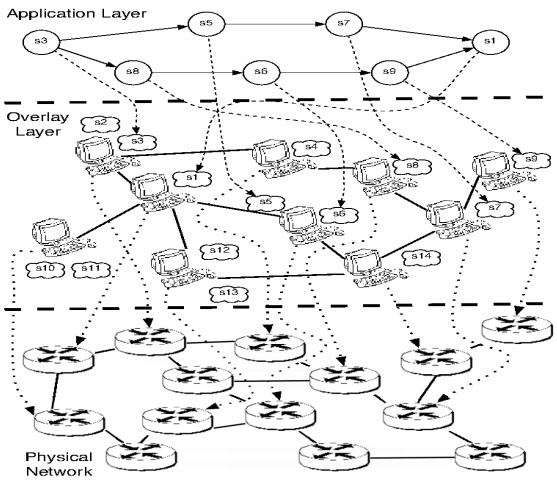

Figure 1. Our large-scale service overlay architecture.

We assume that peers are organized in groups and in section 4 we show that balancing the load among the peers of each group performs comparably to balancing the load among all peers of the network. Peers can be grouped according to their geographical proximity, their network proximity, semantically, or even randomly. The organization of the network topology is outside the scope of this paper and several solutions have already been proposed [14, 17]. Similar to those, we assume that peers are grouped using some criterion and one or more peers in each group are responsible for resource allocation, as discussed in Section 3.

2.2 Application Service Graph

Distributed stream processing applications are modeled as sequences of invocations of services, which are executed in multiple nodes in the overlay network. We use a directed acyclic graph, which we call *application service graph*, to map a distributed application to the overlay network. The vertices of the application service graph represent the services being invoked at a set of peers to accomplish the application execution, while the edges represent connections between those peers. An edge connects two vertices v_i and v_j iff the output of the service corresponding to v_i is the input for the service corresponding to v_j.

Services describe and encode functionalities that are performed at peers. Each service s_i has a name, code, input parameters and output parameters. In order for a sequence of services s_i, s_j to be invoked for the execution of a task, the output of service s_i should be directed to the input of service s_j, provided that they have the same parameters. Then s_i and s_j can be composed to form task $s_i \dot{} s_j$. This composition mechanism can be used to build more complex systems and behaviors.

The application service graph is the outcome of the resource allocation algorithm, when an application execution request arrives. It is composed dynamically at run-time based on the application QoS requirements and the availability of the system resources and stored in all peers participating in the particular application.

The QoS of a service can be approximated using m discrete levels $Q : \{q_1, \ldots, q_m\}$. The resource requirements of a service s_i that provides a quality level q_j are $\{c_i(q_j), b_i(q_j)\}$, where c_i and b_i are the processor cycle and bandwidth requirements, respectively. Usually a higher QoS level means higher requirements in CPU cycles and bandwidth. The resource requirements can be approximately derived from the QoS requirements of the user using for example profiling [16].

3 Adaptive QoS-Aware Resource Allocation

The operation of our QoS-aware resource allocation approach consists of two steps: i) Selecting peers with available resources that offer the requested services to compose the application service graph, so that the QoS demands of the application are satisfied and fair allocation on the system resources on those peers is achieved (3.1). ii) Monitoring the resource usage and application behavior at run-time and dynamically adjusting the quality levels of the tasks, to improve their latencies or to react to changes in the behavior of the applications or the utilization of the system resources (3.2).

3.1 Resource Allocation

We describe the metric for fair load distribution in section 3.1.1, the data structure employed by our resource allocation algorithm in section 3.1.2 and finally the algorithm itself in section 3.1.3.

3.1.1 Fairness Index

The objective of the resource allocation algorithm is to equally distribute the load among the peers of the large-scale overlay. Because the allocation decisions are made by individual peers without global knowledge of the system and the resource loads, we need a metric that captures well the degree of uniformity of the load distribution and does not depend on scaling and magnitude factors. This metric is the *Fairness Index* [13].

Definition 1. *Let $P = \{p_1, p_2, \ldots p_n\}$ be a set of n peers. Let $l_i \geq 0$ be the load of peer i, $1 \leq i \leq n$. The Fairness Index of the load distribution $\bar{l} = \{l_1, l_2, \ldots l_n\}$ among the peers in P is defined as:*

$$\mathcal{F}(\bar{l}) = \frac{\left(\sum_{i=1}^{n} l_i\right)^2}{n \cdot \sum_{i=1}^{n} l_i^2} \quad (1)$$

The Fairness Index has some useful properties that make it an ideal metric to evaluate the fairness of a load distribution. Its values are bounded in the interval $(0, 1]$, and are proportional to the uniformity of the distribution. It is population size and metric independent, it can be applied to any number of nodes and the unit of measurement does not matter. Thus, the Fairness Index can be applied directly to a large-scale overlay, no matter how many peers it incorporates or how loaded they are.

3.1.2 Service Composition Graph

The *service composition graph*, allows us to illustrate all the possible application service compositions in a group of peers and then select the one that best meets the objective of the resource allocation algorithm. The vertices of the service composition graph represent specifications of service inputs or outputs, while the edges represent services offered by peers in the group. An edge connects two vertices v_i and v_j iff there exists a peer that offers a service that takes input of the form specified by v_i and produces an output of the form specified by v_j.

The service composition graph is stored at the peers of a group that run the resource allocation algorithm and

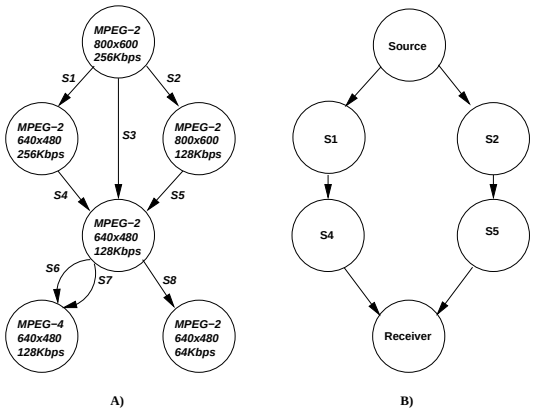

Figure 2. An example of a service composition graph (A) and of the produced application service graph (B).

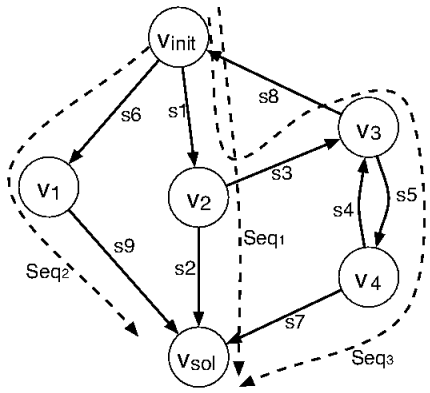

Figure 3. Example of a service composition graph.

is used by the resource allocation algorithm (1) to compose new applications, or (2) periodically to improve the latency of the tasks or to react to processor or resource faults. It refers just to the services that are available in one peer group and is therefore of bounded complexity. It is composed gradually, as peers and services are added to a group and the new connections are detected.

For example, figure 2(A) shows a service composition graph for a system that offers transcoding services. The transcoding quality level and the corresponding resource requirements can be tuned by selecting different implementations for the transcoding functions. Let us assume a user is interested in receiving a video in MPEG-2, 640x480, 128Kbps. Yet, the video is provided by the source in MPEG-2, 800x600, 256Kbps. One can see that we can follow three different paths to go from <MPEG-2, 800x600, 256Kbps> to <MPEG-2, 640x480, 128Kbps>. The choice of which path to use depends on the current resource conditions. Figure 2(B) shows an application service graph that could be produced by this service composition graph.

3.1.3 Resource Allocation Algorithm

The procedure followed by the resource allocation algorithm is the following: First, it finds all paths representing the desirable application using the service composition graph. For each of the paths, it determines whether the quality of service requirements are met. Only the paths representing a solution with which these requirements are met, are considered. Then, the algorithm determines the candidate application service graphs to construct, by mapping each of the paths to the overlay network. For each of the application service graphs, the resulting Fairness Index is estimated. Finally, it selects the service graph for which the Fairness Index is maximized.

Let us assume a service composition graph G_c, as shown in Figure 3. Let v_{init} be the application composition request given by the user (*i.e.,* specifications of the data to be processed) and v_{sol} represent the solution that satisfies the QoS requirements of the user (*i.e.,* specifications of the output data). Each possible solution to the resource allocation problem can be represented as a path Seq_i from v_{init} to v_{sol}. Essentially the path represents a service composition sequence where each edge s_k in the path represents the invocation of a service. If there are multiple possible paths from v_{init} to v_{sol} that satisfy the user's QoS requirements, our goal is to select the one for which the fairest distribution of the load among the peers is achieved. For example, the graph in Figure 3 shows that there are three possible service invocation sequences (paths in G_c) $Seq_1 = \{s_1, s_2\}$, $Seq_2 = \{s_6, s_9\}$ and $Seq_3 = \{s_1, s_3, s_5, s_7\}$. If any of the possible service invocation sequences does not meet the QoS requirements due to processing or bandwidth limitations, these sequences will be ignored.

For each service invocation sequence , the algorithm evaluates the Fairness Index for the corresponding application service graph. This is achieved by calculating the contribution to the load on processor p made by the allocation of service s_j. The service load is derived from the size and specifications of the service input and output, the mean processing time and bandwidth required for the invocation of the service, and the quality of service requirements set by the user for the service. The algorithm then selects the sequence Seq_i that results in the maximum Fairness Index value.

The algorithm identifies the best solution but works in exponential time, because it evaluates all possible so-

Simulation Time	200 seconds
Receiver Buffer Time	2 seconds
Quality Interval Size	3
Media Object Bitrate	(200, 250, 300, 350, 400) Kb
Req. Streaming Bitrate	(50, 75, 100, 125, 150) Kb
# of Total Nodes	1000
# of Source Nodes	400
# of Transcoder Nodes	80
# of Peer Groups (avg.)	10
# of Streams	70
# of Substreams	232

Table 1. Simulation parameters.

lutions. To improve the running time, (1) we avoid examining unacceptable solutions, by constantly checking the QoS, (2) we avoid looping through already examined outputs, and (3) we return as soon as a Fairness Index value greater than a threshold ϵ is found.

3.2 Coordinated Quality Adaptation

Our resource allocation algorithm composes services in such a way that the user QoS demands and their requirements in processor cycles and bandwidth can be accommodated. However, in a dynamic environment the resource loads of the nodes can vary at run-time, degrading the performance of existing tasks. Therefore, adaptation mechanisms that gracefully adapt to changes in the resource usage or the needs of the applications are required. Our approach [5] is to employ a quality adaptation mechanism, which trades off service quality level with resource usage. Locally adapting the quality of a service, might result in quality fluctuations of the end result. Therefore, we employ coordination of the local adaptation decisions based on feedback generated by the receiver of a composite service. Coordinated quality adaptation is triggered when the load on a peer or the latency of a task is too high, or when the user QoS requirements change at run-time. In such cases, reassignment of a task to resources and thus reconstruction of its application service graph might also be needed and can take place by running the resource allocation algorithm again. The cost of the application service graph recomposition is amortized over many application executions.

4 Experimental Evaluation

4.1 Experimental Setup

To evaluate the performance of our resource allocation techniques, we implemented in C++ a simulator for a media streaming and transcoding application over an overlay network. The underlying network topology we used was generated with GT-ITM [22], consisted of 1476 routers, and had an approximate diameter of 750ms. Overlay nodes were randomly attached to different routers. Media sources had a connection bandwidth between 50Kb and 200Kb, while media transcoders had a connection bandwidth between 400Kb and 2Mb and a processing capability between 400M and 800M cycles per second. To take into account the fact that the resources of the system are not dedicated, we randomly added a fluctuating percentage (up to 20%) of extraneous load in the processors and of cross traffic in the network, throughout the execution of the experiment.

We simulated a transcoding application (bit-reduction), in which the execution time of an operation was proportional to the size of a media unit. Data transformation operations on independent media units (*i.e.* groups-of-pictures of MPEG streams) were considered to be the services that needed to be allocated to peers. We simulated 10 levels of output quality for all streams and utilized the same linear utility function that was proportional to the output quality level. The resource allocation algorithm was executed each time an application request arrived in a peer group, while the local adaptation algorithm was executed every second.

We compared our adaptive and *fair* resource allocation algorithm with threshold $\epsilon = 0.8$ against a *random*, a *greedy*, and an *optimal* allocation algorithm. The random algorithm assigned service requests to transcoders blindly. The greedy algorithm searched among the list of the transcoders of a peer group to assign a request to a transcoder that can offer the required bandwidth and processor cycles and returned the first solution it found. The optimal algorithm assigned service requests trying to maximize the fairness of the whole network. Thus, it assumed a central approach, where global knowledge of the loads of all the peers in the network is attainable.

4.2 Results and Analysis

4.2.1 QoS under different loads

In the first set of experiments we investigated the behavior of the resource allocation algorithms, by analyzing the system's performance under different loads. We increased the number of streaming sessions gradually and were thus able to evaluate the scalability of the different algorithms.

Average End-to-end delay. Figure 4 illustrates the average end-to-end delay for all the media units of many streaming sessions. The figure shows that fair resource

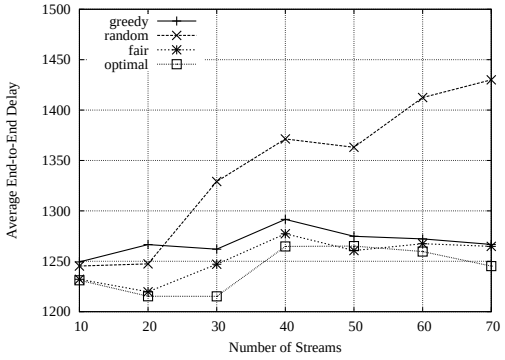

Figure 4. Average end-to-end delay of the media units of all streams vs. the number of streams.

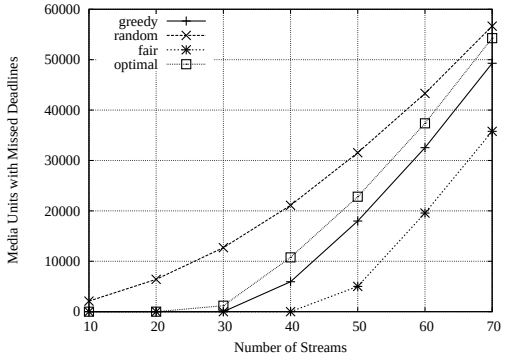

Figure 5. Media units that have missed their deadlines vs. the number of streams.

allocation achieves the lowest end-to-end delay, regardless of the number of streams in the system. Moreover, it maintains a bounded delay, proving that it can offer a scalable solution to the resource allocation problem, as long as the requested resources can be offered by the system. Randomly selecting transcoders results in extreme delays as the load of the system increases. Greedy resource allocation achieves bounded delays as well, since it takes into account the required resources when assigning tasks to the transcoders. Yet, even though the media units arrive with relatively small delays they do not necessarily meet their deadlines, as will be shown in figure 5. It is noteworthy that our fair allocation algorithm achieves average end-to-end delays very close to those that would be achieved by an optimal allocation.

Media units with missed deadlines. Figure 5 shows the degree to which our fair resource allocation algorithm can help the system meet its QoS guarantees (the timing deadlines in particular). Inevitably, as the load becomes more than the system's resources can handle, media units will miss their deadlines. Our fair resource allocation algorithm postpones those negative effects for as long as possible and even then results in considerably less missed deadlines than all other allocation algorithms. As expected, random resource allocation results in missed deadlines even for low loads and extreme numbers of missed deadlines as the load increases. The number of media units with missed deadlines for the greedy allocation algorithm shows that just provisioning for the required resources does not suffice for achieving QoS guarantees, and that a more intelligent load distribution mechanism can have better results. The optimal allocation algorithm results in many missed media units from an unexpectedly low number of streams. This is to show that distributing the streaming and transcoding load fairly across the system is not as efficient as doing the streaming and transcoding locally, within the peer group in which the receiver belongs. This way the streams can reach their destination faster.

4.2.2 System utilization under different loads

In the second set of experiments we observed parameters related to the system utilization. Again, we increased the number of streaming sessions gradually and focused on the scalability of the different algorithms.

Average fairness. As was explained in section 3, how uniformly the load is divided among the processing nodes can affect the efficiency of the system. In figure 6 we present the average of the fairness indices of the individual peer groups, as new streams are admitted in the system. The fair resource allocation algorithm always achieves a more even distribution of the load. As the number of streams is increasing the average fairness increases as well, since tasks can be assigned to all transcoders. When the system becomes overloaded, greedy resource allocation also results in high average fairness, since the load is distributed among all transcoders and the selection of where to place tasks is of no particular importance anymore. Yet, randomly placing tasks, without paying attention to the current load of the nodes, results in low fairness even in those overloaded situations. The fairness of the optimal allocation algorithm is not directly comparable to the rest, since it compares the load distribution among all nodes of the system, instead of averaging the fairness indices of the individual peer groups.

Average load. Figure 7 shows the average of the average loads of each peer group, only for transcoders that have been assigned tasks, as new streams are admitted in the system. This gives us a clear indication of how fairly distributing the load within a peer group can result in lower average load in the whole system. To further explain

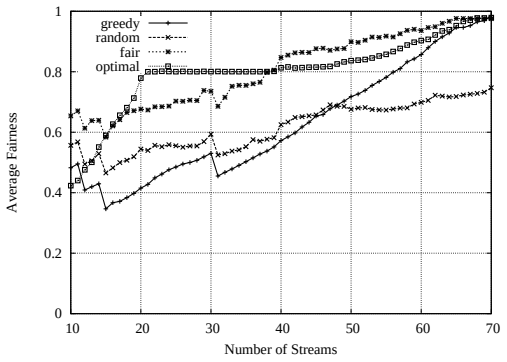

Figure 6. Average fairness of the load distribution across all peers, as a function of the number of streams.

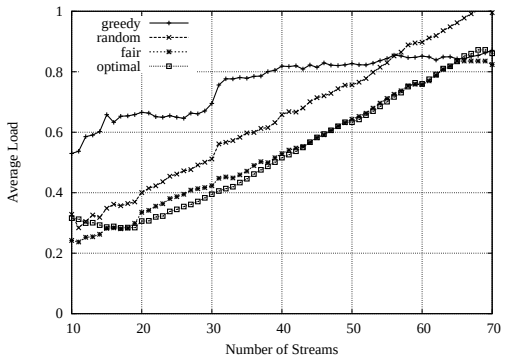

Figure 7. Average load of the system, as a function of the number of streams.

why the average load will not be equal in all situations, suffice it to say that a distribution of tasks in transcoders in peer groups of $[(0.4), (0.4, 0.4)]$ will result in an average load of 0.4, while a distribution of $[(0.8), (0.2, 0.2)]$ will result in an average load of 0.5. Keeping the average load low can be useful, as it will avoid the overloading of certain peers, while other peers remain underloaded. If provisioning for emergency situations is needed, certain nodes can be excluded from the resource allocation procedure, to always be available. The greedy resource allocation algorithm results in average load even higher than the random, since the same nodes are likely to be allocated more tasks, as long as they can accommodate them. Again the average load achieved by the optimal allocation refers to the system as a whole and not to the average of the average loads of each peer group.

5 Related Work

Allocation of processes, tasks or services has been a long studied topic in distributed systems [21, 14, 9]. Recent efforts have studied this topic in distributed stream processing environments. Optimal service composition is accomplished in [9], using a probing protocol and coarse-grained global knowledge. In [21] load variance is minimized by maximizing load correlation. Their approach consists of a centralized algorithm for initial load distribution and a pair-wise algorithm for dynamic load migration. In our work, we assign loads in peer groups and cope with load imbalance using coordinated adaptation. Furthermore, we distribute the load across multiple peers, to eliminate overloaded processors.

Multimedia streaming has been studied extensively in the last few years, mainly from a centralized perspective and without focusing on dynamic user requirements. The importance of providing multimedia applications on overlay networks has only been recognized by recent efforts. These fall in one of the following main categories: (1) Construction of a service graph that satisfies the requested QoS requirements [11].(2) Layered streaming techniques [6] which address the problem of bandwidth heterogeneity by lowering the streaming quality via the omission of stream layers. (3) Proxy-based service provision coordination solutions [12] which make use of a central scheduler. (4) Multicasting mechanisms [2, 18] for efficient content distribution. However, these solutions fail to support multimedia applications that require both communication and processing. More importantly, though, they fail to support multiple media services that can be composed into customized services.

In our previous work we have investigated fair resource allocation for data-sharing applications [7], as well as different aspects of overlays for distributed applications In particular, in [4] we have focused on the task scheduling algorithm, while in [5] we have described a totally decentralized media streaming and transcoding architecture. Our current work builds upon [17] and provides a detailed experimental evaluation.

Similar to our work, in [20] real-time specifications are added to components, while in [19] the authors use an informed branch-and-bound algorithm employing a competence function and forward checking to expedite its execution. These algorithms are centralized and meant to be run off-line. We aim for suboptimal resource allocation, that however is efficient, balanced and can be readjusted at run-time, as the systems we consider are highly dynamic.

The problem of resource allocation with fair quality levels in the context of real time control systems is considered in [10]. The goal is to allocate system resources so that the quality levels of the tasks will be as fairly distributed as possible. A peer's load is defined to be

its CPU utilization factor, while the quantity to be fairly distributed is the quality level offered to each of the running tasks. Our system differs since we try to balance peer load instead of the offered quality level. Also, we consider the problem in the context of wide-area, distributed peer-to-peer overlay networks.

6 Conclusions

In this paper we have proposed a distributed resource allocation algorithm to deal with challenges of heterogeneity and scalability in supporting distributed stream processing applications in large-scale systems. Our approach aims for a fair load distribution among the nodes of the system, while meeting the QoS requirements of the applications. We employ quality adaptation mechanisms to deal with dynamic changes in resource utilization and application behavior The design, implementation and evaluation of our approach is presented. We have evaluated our algorithms by simulating a distributed media streaming and transcoding application. Our results show that a fair and adaptive load distribution in a large-scale system can achieve scalability and maximize the probability of meeting the application requirements.

References

[1] D. Abadi, Y. Ahmad, M. Balazinska, U. C. etintemel, M. Cherniack, J. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The design of the borealis stream processing engine. In *CIDR, Asilomar, CA, USA*, January 2005.

[2] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth multicast in a cooperative environment. In *19th Symposium on Operating Systems Principles (SOSP), NY, USA*, October 2003.

[3] S. Chandrasekaran, O. Cooper, A. Deshpande, M. F. andJ.M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah. Telegraphcq: Continuous dataflow processing for an uncertain world. In *CIDR, Asilomar, CA, USA*, January 2003.

[4] F. Chen and V. Kalogeraki. RUBEN: A technique for scheduling multimedia applications in overlay networks. In *Global Telecommunications Conference (Globecom), Dallas, TX, USA*, November 2004.

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

[6] Y. Cui and K. Nahrstedt. Layered peer-to-peer streaming. In *NOSSDAV, Monterey, CA, USA*, June 2003.

[7] Y. Drougas and V. Kalogeraki. A fair resource allocation algorithm for peer-to-peer overlays. In *8th Global Internet Symposium, Miami, FL, USA*, March 2005.

[8] G. Fry and R. West. Adaptive routing of QoS-constrained media streams over scalable overlay topologies. In *IEEE RTAS, Toronto, Canada*, May 2004.

[9] X. Gu, P. Yu, and K. Nahrstedt. Optimal component composition for scalable stream processing. In *25th International Conference on Distributed Computing Systems (ICDCS), Columbus, OH, USA*, June 2005.

[10] F. Harada, T. Ushio, and Y. Nakamoto. Adaptive resource allocation control with on-line search for fair QoS level. In *IEEE RTAS, Toronto, Canada*, May 2004.

[11] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava. PROMISE: Peer-to-peer media streaming using collectcast. In *11th International Conference on Multimedia, Berkeley, CA, USA*, November 2003.

[12] M. Hicks, A. Nagarajan, and R. Renesse. User-specified adaptive scheduling in a streaming media network. In *OPENARCH, San Francisco, CA, USA*, April 2003.

[13] 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 Equipment Corporation, 1984.

[14] V. Kumar, B. Cooper, Z. Cai, G. Eisenhauer, and K. Schwan. Resource-aware distributed stream management using dynamic overlays. In *25th International Conference on Distributed Computing Systems (ICDCS), Columbus, OH, USA*, June 2005.

[15] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, and J. Hansen. A scalable solution to the multi-resource QoS problem. In *20th Real-Time Systems Symposium (RTSS), Phoenix, AZ, USA*, December 1999.

[16] K. Nahrstedt, D. Wichadakul, and D. Xu. Distributed qos compilation and runtime instantiation. In *IWQoS, Pittsburgh, PA, USA*, June 2000.

[17] T. Repantis, Y. Drougas, and V. Kalogeraki. Adaptive resource management in peer-to-peer middleware. In *13th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS), Denver, CO*, April 2005.

[18] D. Tran, K. Hua, and T. Do. ZIGZAG: An efficient peer-to-peer scheme for media streaming. In *IEEE INFOCOM 2003, San Francisco, CA, USA*, April 2003.

[19] S. Wang, J. Merrick, and K. Shin. Component allocation with multiple resource constraints for large embedded real-time software design. In *IEEE RTAS, Toronto, Canada*, May 2004.

[20] S. Wang, S. Rho, Z. Mai, R. Bettati, and W. Zhao. Real-time component-based systems. In *IEEE RTAS, San Francisco, CA, USA*, March 2005.

[21] Y. Xing, S. Zdonik, and J. Hwang. Dynamic load distribution in the borealis stream processor. In *21st International Conference on Data Engineering (ICDE 2005), Tokyo, Japan*, April 2005.

[22] E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *IEEE INFOCOM 1996, San Francisco, CA, USA*, March 1996.