View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by AIS Electronic Library (AISeL)

Association for Information Systems

AIS Electronic Library (AISeL)

ICEB 2004 Proceedings

International Conference on Electronic Business (ICEB)

Winter 12-5-2004

Distributed Load Balancing in Peer-to-Peer Computing

Shan Zhang

Zheng Qin

Follow this and additional works at: https://aisel.aisnet.org/iceb2004

This material is brought to you by the International Conference on Electronic Business (ICEB) at AIS Electronic Library (AISeL). It has been accepted for inclusion in ICEB 2004 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.

Distributed Load Balancing in Peer-to-Peer Computing

Shan Zhang, Zheng Qin

Software School, Tsinghua University, Beijing, 100084, China

s-zhang@mails.tsinghua.edu.cn, qingzh@tsinghua.edu.cn

ABSTRACT

In this paper, we address the load balancing problem in the context of peer-to-peer computing environments. The key challenge to employ peer-to-peer networks for distributed computing is to exploit the heterogeneous processing capability of the participating hosts as well as the diverse network conditions. The contribution of our work is twofold. First, we model the load balance problem as an optimization problem with the objective of minimizing the system response time. This modeling considers not only the current loading of hosts, but also the fluctuation of network delay, which completely captures the characteristics of the P2P systems. Second, we propose a gradient projection algorithm to solve the optimization problem, which is fully distributed and easy for implementation. Simulation results demonstrate that our scheme has satisfied performance in terms of convergence, response time and load distribution.

Keywords: peer-to-peer computing, load balancing, constrained optimization, gradient projection

1. INTRODUCITON

The Peer-to-Peer (P2P) systems are composed of a set of end hosts that work collaboratly for some common tasks, such as file sharing, on-demand streaming, collaborative computing etc.. They differ from the traditional distributed systems from several aspects. First, the participating hosts in the P2P systems are often distributed across the Internet and higher end-to-end delay is expected between logically closet neighbor nodes. Second, most of these nodes are norml PCs and have diverse processing speed, storage capacity and access bandwidth. These features make it challenging task to design the load balancing scheme to coordinate the distribution of items to be stored, data to be downloaded, or computations to be carried out to the participating hosts so that the heterogeneity of the hosts can be fully exploited.

Addressing this problem, several solutions have been proposed in literature [1]-[4]. These schemes are focused on the load balancing for structured P2P systems. Structured P2P systems employ a class of distributed hash tables (DHT) for item storage and retrieval. The main limitation of this kind of systems is that the distributed hash tables may not produce load balance as good as standard hash function, so there is a $\Theta(\log(N))$ imbalance factor in the number of items stored at a node [2]. In [2], Rao et. al. assume that the system is static, i.e., the membership and data items are stable over a timescale that the load balancing algorithm can be performed. They employ the concept of virtual server that represents a peer in DHT. Each physical node can host one or more virtual servers so that the storage and retrieval of data items are conducted at the virtual server level instead of the physical node level. The key to achieve load balance is then to move virtual servers from overloaded physical nodes to lightly loaded physical nodes. This concept is extended to the dynamic P2P systems by Godfrey et. al. in [4], where data items can be continuously inserted and deleted, and nodes may change their states frequently. They propose to maintain the loading information of the peer nodes on a set of directory servers. These directory servers periodically schedule the reassignment of the virtual servers among the physical nodes that they are serving. However, these solutions do not consider the cost to move the virtual servers between the hosts. In [5], Zhu and Hu proposed a proximity-aware load balancing scheme to address this issue. The basic idea is to utilize the proximity information of the physical nodes, so that the virtual servers are only reassigned and transferred between physically close heavy nodes and light nodes so that the cost of performing reassignments is reduced.

The drawback of the virtual server based load balancing schemes is that running multiple virtual nodes may increase the bandwidth consumption in maintaining the network connectivity across neighbor nodes in the peer-to-peer network. In [6], Karger et. al. address this problem by arranging for each node to activate only one of its virtual nodes at any given time. Specifically, the node checks its inactive virtual nodes periodically. If it finds that the load distribution has changed, it may migrate to one of these virtual nodes. In this way, previous load is shifted to other nodes, and it will taks responsibility for loading of the newly migrated virtual nodes.

All of the aforementioned schemes are based on the structured P2P systems where the load distribution is tightly controlled by the distribution hash functions. Although the virtual server concept is elegant and easy

to implement, the achieved load balancing is ad-hoc in that it relies on the reassignment or migration of data items after they have been assigned. The fineness depends on the space allocated for each virtual server. The tradeoff, however, is not easy to justify. Furthermore, the scheme adopted in [4] essentially resorts to the centralized solution, which makes the directory servers vulnerable to single-point-failure.

In this paper, we consider the load balancing problem in the context of unstructured P2P systems. Example of this kind of systems includes Gnutella, KaZaA, etc. A salient feature of these unstructured P2P systems is that the host can fully control where its data items can be placed, and the amount of the data items that should be assigned to these hosts. This feature is desirable for load balancing purpose since it allows for a fine-grained allocation of load across all participating hosts according to their loading. However, the results is not only limited to unstructured P2P system, it is also applicable for structured p2p system. For example, it can be used to find a finegrained reassignment of virtual servers for this kind of p2p systems.

The contribution of our work is twofold. First, we model the load balance problem as the minimum cost problem. This modeling considers not only the current loading state of the participating hosts, but also the fluctuation of network delay, which completely captures the characteristics of the P2P systems when they are employed for massive computing purpose. Second, we propose a gradient projection algorithm to solve the optimization problem, which is fully distributed and easy for implementation.

In the reset of this paper, we first present the system model and problem formulation in Section 2 and discuss the necessary conditions for the optimization problem. In Section 3, we then propose an algorithm based on the classic gradient projection methods. By identifying the difficulties to implement the algorithm in practice, we propose an approximation solution which is suited for practical implementation. We also discuss the inherit reasonability of our model for the p2p computing environments. Simulation results are presented in Section 4 and finally we conclude this paper in Section 5 and suggest some possible directions for future research.

2. SYSTEM MODEL AND PROBLEM FORMULATION

2.1 Problem Statement

We are given a peer-to-peer system and a set of *N* hosts. Each host $i \in N$ can choose a set of S_i hosts from these *N* hosts as its servers to carry out the computation for its raw data. These servers can be chosen randomly as in the unstructured peer-to-peer networks, or can be determined according to the distributed hash function as that of the structured peer-to-peer networks.

For each host *i*, we are given a scalar r_i referred to as the generating rate of the raw data of host *i* to be computed. In the context of the peer-to-peer computing, r_i (measured in units/second) could be the data rate exporting from host *i* to its servers for computation for the final results. The objective of our problem is to divide r_i of host *i* among the set of S_i servers in a way that the resulting loading distribution across the overall peer-to-peer system minimizes a suitable cost function.

We denote x_{ik} as the data rate that is assigned by host *i* to its server $k \in S_i$. The collection of assignment x_{ik} must satisfy the following constraints

$$\sum_{k \in S_i} x_{ik} = r_i, \quad \forall i \in N \tag{1}$$

$$x_{ik} \geq 0, \quad \forall k \in S_i, i \in N \tag{2}$$

For peer-to-peer computing, the primary design objective is to minimize the response time for the tasks, which is the duration from the time that a host assigns the raw data to the server to the time that the final result is computed and retrieved. In peer-to-peer systems, the participating hosts have highly diverse processing speeds and are widely distributed across the Internet. As a result, the response time is composed two components, one is host dependent processing delay and the other is network dependent end-to-end delay.

For host-dependent delay, we adopt the response time model from [7] by treating the computing process as an M/M/1 system. Specifically, suppose a host *k* is chosen by a set of A_k hosts as their servers. Then the aggregated data rate to be carried by this host is

$$J_k = \sum_{i \in A_k} x_{ik} \tag{3}$$

Suppose the host *k* has a processing speed σ_k, then the cost function of the processing delay for each host *i* to get the computation result from this host can be obtained following M/M/1 queuing model as follows

$$D_{ik}^h(x_{ik}) = \frac{J_k}{\sigma_k - J_k} \tag{4}$$

For network delay, it consists of the time for delivering the raw data from the host to the server plus the time for retrieving the results from the server. Without losing generality, we can assume the volume of the raw data is very larger than that of the results, so the delay of getting the results can be ignored and we only consider the former one. Assume the path P_{ik} between a node *i* and node *k* consists of a set of links of $l_0, l_1, \cdots$. Then the cost function between *i* and *k* can be represented by a superposition of delays of all links along the path by modeling each link as M/M/1 queue as follows

$$D_{ik}^n(x_{ik}) = \sum_{l \in P_{ik}} \left[\frac{Y_l}{C_l - Y_l} + d_l Y_l \right] \tag{5}$$

where Y_l is the aggregated traffic over link *l* including x_{ik}, C_l and d_l are the link capacity and propagation delay of link *l* respectively.

Summing up, the overall cost function for response time between host *i* and *k* is

$$D_{ik} = D_{ik}^h + D_{ik}^n \tag{6}$$

Consider the overall cost function of the form

$$\sum_{i\in N}\sum_{k\in A_i} D_{ik}(x_{ik}) \tag{7}$$

The problem is to find a set of job assignments $\{x_{ik}\}$ that minimize the cost function (6) subject to the constraints of equations (1)-(3). More formally, the load balancing problem can be stated as follows

$$\begin{aligned} &\text{minimize} \quad D(x) \\ &\text{subject to} \quad \sum_{k\in S_i} x_{ik} = r_i, \quad \forall i \in N \\ &\qquad\qquad x_{ik} \ge 0, \quad \forall k \in S_i, i \in N \end{aligned} \tag{8}$$

where

$$\begin{aligned} D(x) &= \sum_{i\in N}\sum_{k\in S_i} D_{ik}(x_{ik}) \\ &= \sum_{i\in N}\sum_{k\in S_i} \left(D_{ik}^h(x_{ik}) + D_{ik}^n(x_{ik}) \right) \end{aligned} \tag{9}$$

2.2 Necessary Conditions for Optimality

We first calculate the partial derivative of the objective cost function (9) with respect to x_{ik}, which is

$$\frac{\partial D}{\partial x_{ik}}(x) = D_{ik}^h{}' + D_{ik}^n{}' \tag{10}$$

where $D_{ik}^h{}'$ and $D_{ik}^n{}'$ are the derivatives of $D_{ik}^h(x)$ and $D_{ik}^n(x)$ respectively. Equation (10) can be interpreted as the overall response time between host *i* and server *k* if the server's processing delay is taken to be the first derivative $D_{ik}^h{}'$ and the network delay to be the first derivative $D_{ik}^n{}'$.

According to the Kuhn-Tucker theorem(see also [9]), at optimality, a host only assigns its jobs over servers that have the minimum response time given by (10). More specifically, let $x^* = \{x_{ik}^*\}$ be an optimal assignment vector. Then if $x_{ik}^* > 0$ for some $k \in S_i$, we have[9]

$$\frac{\partial D}{\partial x_{ik'}}(x^*) \ge \frac{\partial D}{\partial x_{ik}}(x^*), \quad \forall k' \in S_i \tag{11}$$

That is, the optimal assignment is *positive* only on servers with the minimum first derivative response time.

3. LOAD BALANCING ALGORITHM AND DISTRIBUTED IMPLEMENTATION

3.1 Gradient Projection Methods

The standard technique to solve the equality constrained optimization problem (8) is the *gradient projection algorithm* [9]. The basic idea is to iteratively adjust the amount of assignment in opposite to the gradient of the objective function and projected to feasible space specified by the constraint functions. In general, the gradient projection algorithm takes the iterative form as follows

$$x^{t+1} = \left[x^t - \alpha^t \nabla D(x^t) \right]^+, \quad t = 0,1,\cdots \tag{12}$$

where $\alpha^t > 0$ is the step size at time *t*, $\nabla D(x^t)$ is a vector with the *(i,k)*th element to be the first derivative of $D(x)$ with respect to x_{ik}, which is $\frac{\partial D}{\partial x_{ik}}(x)$ as shown in (10) at time *t*, and for any vector z, the function $[z]^+$ denotes the projection of z onto the feasible space.

The iterative algorithm in (11) can be expressed individually for each host-server pair as the following

$$x_i^{t+1} = \left[x_i^t - \alpha^t \nabla D_i(x^t) \right]^+ \tag{13}$$

where $x_i^t = \left\{ x_{ik}^t, k \in S_i \right\}$ is the assignment vector of host *i* over all its servers at time *t*.

3.2 Practical Implementation

The projection algorithm in (13) can be carried out individually by each host according to the locally received cost function information, so it is well suited for distributed implementation. The most straightforward approach is for each host to broadcast the value of the current total aggregated loading J_k to all hosts that select it as the server. Each host then computes the host-dependent cost D_{ik}^h and its derivative. The difficult here, however, is that the network-dependent cost D_{ik}^n can only be implicitly obtained since in general it is not practical to ask intermediate network nodes for detailed traffic information.

Notice that the parameters involved in the projection algorithm are only the first derivative $D_{ik}^h{}'$ and $D_{ik}^n{}'$, which, as we have discussed, can be interpreted as the response time during the host and the network respectively. For host-dependent response time, it is actually the time for waiting for the computation results, while for network-dependent response time, it is actually the end-to-end delay between the host and server. So we can actually directly measure these two

values and use them to approximate the first derivative of two cost functions.

Let $\tilde{d}_k$ be the measured processing delay at host *k* and $\tilde{d}_{ik}$ be the network delay between host *i* and *k*. Here $\tilde{d}_{ik}$ can be measured by host *i* itself, while $\tilde{d}_k$ can be measured and fed back by host *k*. Host *i* receiving all these information from all its servers can determine the optimal assignment of its task among these servers using the iterative algorithm as follows

(1) $\tilde{d}_{i,\min} \leftarrow \min\{\tilde{d}_{ik} + \tilde{d}_k \mid k \in S_i\}$;

(2) $k_0 = \arg\min\{\tilde{d}_{ik} + \tilde{d}_k \mid k \in S\}$;

(3) **foreach** $k \in S_i$, **do**

(4) $\Delta_{ik} \leftarrow \tilde{d}_{ik} + \tilde{d}_k - \tilde{d}_{i,\min}$;

(5) **if** $k \neq k_0$, **do**

(6) $x_{ik} \leftarrow \max\{0, x_{ik} - \alpha \times \Delta_{ik}\}$;

(7) **end**

(8) **end**

(9) $x_{ik_0} \leftarrow r_i - \sum_{k \in S_i, k \neq k_0} x_{ik}$

In this algorithm, k_0 is the index of the server with the minimum total response time $d_{i,\min}$. In step (4), the algorithm computes the difference of the response time between any server and this minimum one. In step (6), it decreases the assignment to each non-minimum server in proportion to this difference and projection to the feasible space. The max projection function guarantees that nonminimum with zero assignment originally always stays at zero. In step (9), it simply assigns the rest task to the server with the minimum response time. It is trivial to prove that the assignment to the minimum response time server is always increased, while the nonminimum servers are always reduced until to zero, eventually only these severs with minimum (and identical) response time have positive assignment as (11) indicates. This guarantees that the assignments approach to optimality eventually.

3.2 Discussion

Unlike traditional distributed computing system, the participating hosts in the peer-to-peer network may be distributed across wide area network, which make the network delay a non-negligible factor. At the same time, computers involved in the p2p computing are normally personal computers that have diverse processing speeds. The processing delay is the major concern for the p2p computing. Our model incorporates both factors into the same framework. Therefore the load balance problem exhibits distinct features from the traditional distributed computing as well as other delay-insensitive p2p systems. Depending on the relation of the network delay and processing delay, we can envision the following two scenarios;

- If the processing delay is the dominate part of the overall response time, i.e., process delay is far more longer than network delay, our model is close to the traditional load balance problem where the objective is to assign the jobs across the servers according to their processing capability.
- If the network delay is the dominate part of the overall response time, our model is close to the optimal routing problem. The objective here becomes to assign the tasks across the servers to avoid overloading the network connections with longer delay.

Clearly, under this framework, in contrast to the conventional wisdom that a host should choose these physically closet nodes as the servers, which may led to task concentration over these nodes and a higher processing delay is expected. Instead, a tradeoff should be achieved to between the processing delay and network delay, which makes the selection of the server a rather challenging problem.

4. PERFORMANCE EVALUATION

In this section, we evaluate the distributed load balancing scheme proposed in this paper using simulations based on a randomly generated network. The objective is to investigate the convergence of the distributed implementation, the effects of the number of servers for each host on the performance of the scheme, and the behavior of the scheme under different composition of the network delay and processing delay. The network used in the simulations has 100 nodes, among of them, 10 nodes are randomly selected as the hosts which generate the raw data at a rate of 10k bits per seconds. The rest nodes serve as the servers for these hosts. The number of servers for all hosts is the same and the map of host and server is randomly determined.

Fig. 1 illustrates the convergence of the mean response time of these 10 hosts within a 200-seconds run of simulation. In this simulation, each host randomly selects 5 servers and distributes its task evenly across its servers at the beginning, which are far from optimal as shown in the figure: all of them incur higher response time which is the mean value of the response time to all the servers of each host. The response time of all hosts drops quickly within the first 20 seconds of the simulation and converges to the stable value after the simulation is run for about 30 seconds. The result suggests that proposed algorithm can reduce the tasks from the overloaded servers or servers with congested network connections and distribute them to the under-utilized servers or these that have good network conditions.

Fig. 2 compares the mean response time of all hosts under different number of servers for each host. We consider five cases where the number of servers of each hosts varies from 2 to 10. As this figure shows, the hosts incur a very higher response time when they have only two servers for each. This is reasonable because in this case a host has little freedom to adjust the assignment of the task if both of its two servers are overloaded or under severe network conditions. The response time is significantly reduced as the number of the servers of each host is increased to four as well as from four to six. However, the improvement of the response time is slowed down as the number of servers is further increased, and the convergence of the response time has a subtle increment when the number of servers is increases. This simulation suggests that the server number should be chosen at a reasonable value so to achieve the tradeoff between the response time and the convergence time.

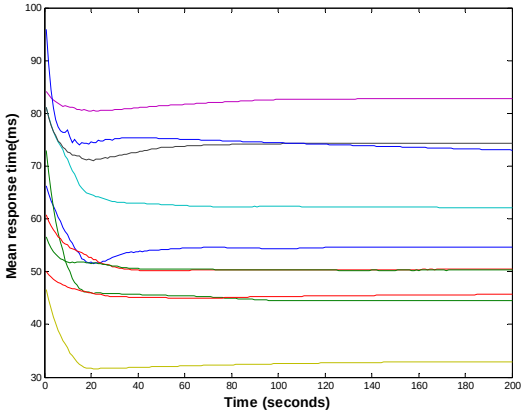

Fig. 1 Convergence of response time vs simulation time

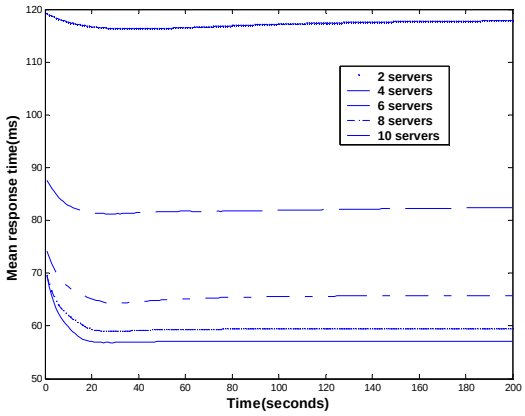

Fig. 2 Response time under different number of servers

As discussed in the previous section, our model tries to capture two factors that affect the performance of p2p computing environments. The proposed scheme may exhibit different properties depending on the composition of the overall response time. In this simulation, we will verify this characteristic using two set of settings. We use the same network topology as above simulations so that the average network delay is roughly fixed. We then adjust the processing speeds of the nodes so that in the first case, the ratio of the processing delay to the network delay is about 5:1, while in the second setting, this ratio is about 1:5. In Fig. 3 and Fig. 4, we illustrate the throughput (the message processed in bits per seconds) of the host under these two settings respectively. Each figure shows the max, min and mean throughput of all nodes. As shown in Fig.3, since in this case the processing delay is very larger comparing to network delay, the algorithm behaviors like the normal load balancing scheme that can distribute tasks as evenly as possible overall servers. So the max, min and mean value are very closer. While as shown in Fig. 4, here the network delay is comparable to processing delay, so the algorithm considers not only the server load but also the network delay, so the server throughput is not distributed evenly, however, the overall response time still approaches to the optimality not shown here.

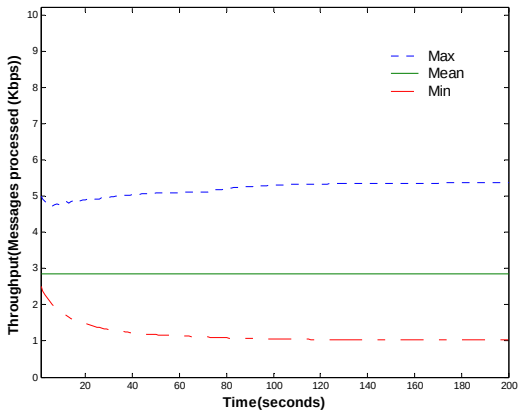

Fig. 3 Throughput under first setting

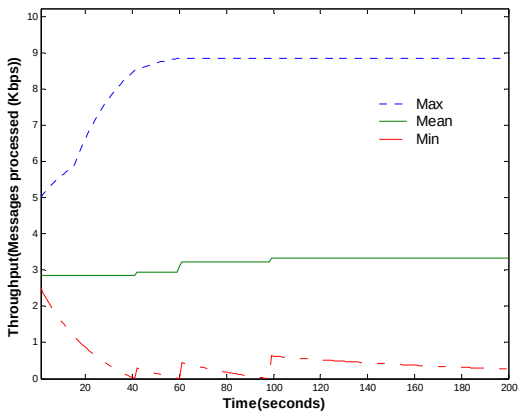

Fig. 4 Throughput under second setting

5. CONCLUSION AND FUTURE WORK

In this paper, we have presented the modeling and solution for the loading balancing problem for a p2p computing system. In this system, the primary challenge is to allocate the tasks overall all participating nodes so that the overall response time is minimized. We model

this as a constrained optimization problem. The model captures the distinct features of p2p computing, that is, the cost involved in the computation includes not only the processing delay, but also the network delay. Leveraging some well studied analysis techniques, we can solve the constrained optimization problem using the gradient projection algorithm, which is well suitable for distributed implementation. Using extensive simulation experiments, we evaluate the proposed scheme in terms of convergence, response time as well as the tradeoff the processing delay and network delay. Experimental results demonstrate our scheme has satisfied performance.

In this study, we have only considered the static situation where the nodes are stable in the network, and a host never changes its servers once selected. It is interesting to investigate the case where the node can join or leave the p2p networks dynamically. A leaving node may broke the ongoing computing, this lead to another problem that a host may have to change its servers for better performance or for failure of the existing servers. It is interesting to investigate the stability of the proposed scheme in this dynamic environment, which will be the focus of the further work.

REFERENCES

[1] I. Stoica, R. Morris, David Karger, M. Frans Kaashoek,and H. Balakrishnan, “Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications,” in *Proc. ACM SIGCOM*M, San Diego, 2001, pp. 149–160.

[2] A. Rao, K, Lakshminarayanan, S. Surana, R, Karp, and I. Stoica, “Load Balancing in Structured P2P Systems,” in *Proc. IPTP*S’03, Feb. 2003.

[3] J. Byers, J. Considine, and Michael Mitzenmacher, “Simple Load Balancing for Distributed Hash Tables,” in *Proc. IPTP*S’03, Feb. 2003.

[4] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp and I. Stoica, “Load Balancing in Dynamic Structured P2P Systems”, *Infocom’04*, March, 2004.

[5] Yingwu Zhu and Yiming Hu. "Towards Efficient Load Balancing in Structured P2P Systems". *IPDPS'04* . April 2004.

[6] David R. Karger and Matthias Ruhl, “Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems”, *IPTPS'04* . Feb. 2004.

[7] Subhash Suri, Csaba Toth and Yunhong Zhou. “Uncoordinated Load Balancing and Congestion Games in P2P Systems”, *IPTPS'04* . Feb. 2004.

[8] Dimitri P. Bertsekas and John N. Tsitsiklis, “Parallel and Distributed Computation: Numerical Methods”, Athena Scientific, 1997.

[9] Dimitri P. Bertsekas and R. Gallager, “Data Networks”, Printice-Hall Inc. 2nd edition, 1992.