Optimal Client-Server Assignment for Internet Distributed Systems

LAILA FATIMA¹, T. MALATHI²

¹PG Scholar, Dept of ECE, Aurora Scientific Technological and Research Academy, India, E-mail: decentlaila_786@yahoo.com.
²Asst Prof, Dept of ECE, Aurora Scientific Technological and Research Academy, India.

Abstract: We investigate an underlying mathematical model and algorithms for optimizing the performance of a class of distributed systems over the Internet. Such a system consists of a large number of clients who communicate with each other indirectly via a number of intermediate servers. Optimizing the overall performance of such a system then can be formulated as a client-server assignment problem whose aim is to assign the clients to the servers in such a way to satisfy some pre-specified requirements on the communication cost and load balancing. We show that 1) the total communication load and load balancing are two opposing metrics, and consequently, their tradeoff is inherent in this class of distributed systems; 2) in general, finding the optimal client-server assignment for some pre-specified requirements on the total load and load balancing is NP-hard, and therefore; 3) we propose a heuristic via relaxed convex optimization for finding the approximate solution. Our simulation results indicate that the proposed algorithm produces superior performance than other heuristics, including the popular Normalized Cuts algorithm.

Keywords: Distributed Systems, Client-Server Systems, Graph Clustering, Load Balancing, Communication Overhead, Optimization.

I. INTRODUCTION

An internet distributed system consists of a number of nodes (e.g., computers) that are linked together in ways that allow them to share resources and computation. An ideal distributed system is completely decentralized, and that every node is given equal responsibility and no node is more computational or resource powerful than any other. However, for many real-world applications, such a system often has a low performance due to a significant cost of coordinating the nodes in a completely distributed manner. In practice, a typical distributed system consists of a mix of servers and clients. The servers are more computational and resource powerful than the clients. A classical example of such systems is e-mail. When a client A sends an e-mail to another client B, A does not send the e-mail directly to B. Instead, A sends its message to its e-mail server which has been previously assigned to handle all the e-mails to and from A. This server relays A’s e-mail to another server which has been previously assigned to handle e-mails for B. B then reads A’s e-mail by downloading the e-mail from its server. Importantly, the e-mail servers communicate with each other on behalf of their clients.

The main advantage of this architecture is specialization, in the sense that the powerful dedicated e-mail servers release their clients from the responsibility associated with many tasks including processing and storing e-mails, and thus making e-mail applications more scalable. E-mail systems assign clients based primarily on the organizations that the clients belong to. Two employees working for the same company are likely to have their e-mail accounts assigned to the same e-mail server. Thus, the client server assignment is trivial. A more interesting scenario is the Instant Messaging System (IMS). An IMS allows real time text-based communication between two or more participants over the Internet. Each IMS client is associated with an IMS server which handles all the instant messages for its clients. Similar to e-mail servers, IMS servers relay instant messages to each other on behalf on their clients. In an IMS that uses the XMPP (Jabber) [2] protocol such as Google Talk, clients can be assigned to servers independent of their organizations. Furthermore, the client-server assignment can be made dynamic as deemed suitable, and thus making this problem much more interesting.

In the XMPP, a username is set as user@domain (e.g., nishida@jabber.org) just like an e-mail account, where domain usually stands for a server name in which user is registered. When a user aaa@domain sends a message to another user in the same domain bbb@domain, the message is delivered only through the domain server, i.e., aaa → domain server → bbb. The clients do not directly exchange their messages each other. When a user aaa@domain1 sends a message to another user in a different domain bbb@domain2, the message is sent as: aaa → domain1 server → domain2 server → bbb. This design is indeed simple and scalable. If the number of users increases, another server can be added to accommodate the new users. Herein, we consider server load in an IMS. We assume all communications are encrypted. The amount of load on a server (we call it
communication load) is substantially proportional to the amount of data that the server receives (= r) for the following reasons: The server basically sends the same amount of data (= r) to a client/another server.

![Diagram](image)

**Fig. 1. Example of client assignment to servers.**

The processing times taken for decrypting the received data and for encrypting the sending data are both proportional to r. Except for the encryption and decryption, the load on the server is dominated by copying the data among a network device, the operating system’s kernel, an IMS program and sometimes a hard drive, which is also proportional to r. Based on these, we need to consider how to optimally assign clients to servers, beginning with the following observations: Suppose both clients i and j are assigned to server 1 and i sends a message of size 1 to j, then the message is sent only via server 1 (see Fig. 1a). We define the amount of communication load on server 1 in this case as 1. Suppose client i is assigned to server 1 and j is assigned to server 2. If i sends a message of size 1 to j, then the message is delivered through servers 1 and 2 (see Fig. 1b). The amount of communication load on server 1 is still 1 and that on server 2 is also 1, because both server 1 and 2 need to process the message of size 1. (Note we assume a system always consists of homogeneous machines in this paper.)

From the above two cases, we know that assigning clients to different servers doubles the amount of total communication load compared to assigning them to the same server. Hence, we need to assign clients to servers so that the amount of total communication load is minimized. If two clients who exchange many messages with each other are assigned to two different servers, then the amount of total communication load increases. On the other hand, if two clients who never exchange messages are assigned to different servers, then the amount of total communication load stay unchanged. So, it makes sense to assign clients that exchange many messages to the same server and to assign clients that exchange few messages to different servers in terms of minimizing the overall communication load. Since we use multiple servers, we also need to balance the communication load among the servers for the following reasons:

- As a heavily loaded server typically exhibits a low performance, we would like to avoid the situation.

- If one server is overloaded, we need to add another server to distribute the load, which is economically inefficient and usually increases the overall communication load (see above). For instance, if the loads on server 1 and 2 are 1.2 (i.e., 20 percent overloaded) and 0.6, respectively, then we have to add a server to reduce the load on server 1 to less than 1.0. However, if the loads are 0.9 and 0.9, then there is no need to do that.

- To minimize the amount of total communication load, assigning all clients to one server is optimal. However, it is impossible due to overloading and completely loses the load balance. Simple load balancing does not usually take account of reducing the overall communication load.

Given the observations above, we must strike a balance between reducing the overall communication load and increasing the load fairness among the servers, i.e., the load balance. The primary contribution of this paper is a heuristic algorithm via relaxed convex optimization that takes a given communication pattern among the clients as an input, and produces an approximately optimal client-server assignment for a pre-specified tradeoff between load balance and communication cost. Next, we describe a number of emerging applications that have the potential to benefit from the client-server assignment problem.

### A. Emerging Applications

The client-server assignment problem is also relevant to a host of emerging applications ranging from social network applications such as Face book and Twitter to online distributed auction systems such as eBay. Face book is a system that allows circles of friends to exchange messages and pictures among themselves. Since friends are likely to communicate with each other than non friends, assigning friends to the same server will reduce the inter server communication and will result in reducing the overall communication load. At the same time, it is preferable to balance the communication load. This is exactly the client server assignment problem encountered in the IMS. Online distributed auction systems are another candidate for applying the client-server assignment. If a user logged in a server which has contents that are mostly not of interest to the user, then on average, every item search by a user will generate a larger communication overhead, as the search must be done across multiple servers therefore, letting a user log in the server that is likely to have contents of interest to a user will raise the efficiency. In this case, the types of contents can also be viewed as clients.

The client-server assignment also has the potential to be applicable to distributed database systems, such as Map Reduce [3]. Assigning the search keywords which are often queried together to the same servers will reduce the inter server communication. In this case, the search keywords correspond to the clients in the above IMS. Note that we are not focused on real-time (or highly dynamic) client-server assignment in this paper because of the relatively expensive
Optimal Client-Server Assignment for Internet Distributed Systems

Computation cost. Instead, moderately dynamic applications such as social networks where users do not change their friends too frequently are our targets (fig 2). We assume that our algorithms are used in such situation that the recalculation of assignment is needed only periodically, e.g., once a week.

![Graph Image]

Fig.2. Example of bi-partitioning.

II. CLIENT SERVER ASSIGNMENT PROBLEMS

Client Server assignment problem can be approached based upon various factors. There has been lot of research in this area.

A. Clustering Algorithms

Client Server assignment problem is an instance of clustering problem. Clients and their communication pattern can be denoted as a graph, with vertices representing the clients and the edges between two vertices representing the communication between the respective clients. Communication frequency between two clients can be represented by the weight of the edge between the corresponding vertices. Clustering algorithm forms fixed number of clusters of clients based on a given objective. Objective of the clustering algorithm in this paper is to minimize the amount of inter group communication and balance the sum of weights of all edges in the group. Normalized cuts (NC) is a clustering algorithm that partitions an undirected graph into two disjoint partitions such that $\text{F_{ncut} = \frac{W_{1,2}}{W_{1,1}+W_{1,2}} + \frac{W_{2,1}}{W_{2,1}+W_{2,2}}}$ is minimized, Where $W_{i,j}$ is the sum of weights of all edges that connects the vertices in group i and j.

In order to solve Fncut efficiently, NC uses Eigen values of adjacency matrix. But NC tends to cause imbalance in the volumes of the groups by isolating the vertices that do not have strong connection to others. Balanced clustering algorithms for power law graphs are examined. Author concludes that combining multiple trials of a randomized flow-based rounding methods and solving a semi definite program yields effective results. In it is shown that the optimal client server assignment for pre-specified requirements on total communication load and load balancing is NP-hard. A heuristic algorithm based on relaxed convex optimization is used for finding the approximate solution to the client server assignment problem

B. Distributed Interactive applications (DIAs)

DIAs are networked systems that allow multiple participants at different locations to interact with each other. Wide spread of client locations in large-scale DIAs often requires geographical distribution of servers to meet the latency requirements of the applications. In the distributed server architecture, client assignment to servers directly affects the network latency involved in the interactions between clients focuses on client assignment problem for enhancing the interactivity performance of DIAs. In this paper, the problem is formulated as a combinational optimization problem on graphs and proved to be NP-complete of assigning clients to appropriate servers. Three heuristic algorithms are proposed, namely, nearest assignment, Greedy assignment, Distributed greedy assignment and are compared and evaluated using real Internet latency data is an improved version and the results show that the proposed algorithms, nearest Assignment algorithm, Modify Assignment and Distributed modify assignment, are efficient and effective in reducing the interaction time between clients. However, both consider the client assignment in DIAs and do not consider the load on the server and other factors.

C. Load-distance balancing Problem (LDB)

Client server assignment is modeled on the fact that the delay incurred by a client is dependent on load of the server and the distance to the assigned server. Delay on the client side is the sum of network delay (proportional to distance to its server) and congestion delay at the server. Hence it focuses on distance between the client and the server and the load on the server in client-server assignment. The problem is defined as Load-distance balancing (LDB) problem. This paper targets 2 flavors of LDB. In the first flavor, the objective is to minimize the maximum incurred delay using an approximation algorithm and an optimal algorithm. In the second flavor, the objective is to minimize the average incurred delay, which the paper mentions it as NP-hard and provides a 2-approximation algorithm and exact algorithm. However to the best of our knowledge there is no clustering algorithm that is designed as a Semi-definite programming problem to achieve our goal: Load balancing on servers and minimizing the communication load between servers, for client-server assignment.

III. MODEL OF COMMUNICATION

To solve this problem of client-server assignment we first design a model for communication. Every client is assigned to a single server. When a client has to interact with another client it passes on message to its assigned server and server forwards that message. If the destination client is associated with same server as source client, message is forwarded directly from the server to destination client. If the destination client is assigned to another server, message is forwarded to destination client server from source client’s server. In fig 3 we have 2 servers, Servers A and B and 3 clients Ci, Cj and Ck. Ci and Cj are connected to the same server, Server A. When client Ci wants to communicate with client Cj, Cj forwards the message to its associated server A. Server A forwards the message to client Cj. In process of message forwarding any server performs only two actions, reception and forwarding. If $r$ is the size of message, then load on the server is increased by $r$ for reception and forwarding.
Fig. 3. Client-server assignment.

As Ci and Cj are assigned to the same server, communication between them increases the load on Server A by r. Consider the communication between clients Ci and Ck. Load on Server A increases by r as Server A receives a message from Ci and forwards it to Ck’s associated server. Load on Server B is also increased by r as Server B also receives and forwards the message to Ck. Therefore, overall load increases by 2r when clients are assigned to two different servers as compared to r in case of a single server. Based on this communication model, we solve the client assignment problem. Initially, we develop the algorithm for client assignment in distributed systems with only 2 servers. Then we will expand the algorithm for n servers.

IV. SIMULATION RESULTS

In this section, we evaluate the performance of the following algorithms abbreviated as follows:

BCO-V: The binary splitting via relaxed convex optimization based.
BCO-E: The binary splitting via relaxed convex optimization based.
NC: The Normalized Cuts
Graclus: An efficient graph clustering algorithm in [9]
RND: The random client-server assignment

We examined the BCO-V and BCO-E for ß = 0.5, 0.3, 0.1, 0.05 and used as a convex optimization solver. For the NC algorithm, we used the Matlab code at, and for the Graclus, we used the version 1.2 codes at. Both programs are written by the authors of their original papers [4], [9]. Note that we ran C-based programs (BCOs, Graclus) with Intel Pentium E2140 machines and ran Matlab-based programs (NC) with High Performance Computing Cluster at Oregon State University. Therefore, the comparison on the computation time is approximate.

A. Examples of Graph Clustering

Fig. 4 and 5 show our simulation results with small graphs. The reason for using small graphs is because it is easy to examine the results of each algorithm in details. Furthermore, it is feasible to find the optimal solution by an exhaustive search, which helps us to quantify how good an approximate solution produced by a heuristic is. In the simulation, ß = 0.5 is used for the BCOs. Also, the results by the NC and Graclus are added for the comparison. F is the metric, and the smaller, the better client-server assignment we have. As shown in the figures, the NC tends to isolate small volumes of groups that do not have strong connection to others. Our BCO methods well balance the total weight of the associated edges in each group, which appears as smaller F values. The Graclus clusters better than the NC but does not better than the BCOs.
Optimal Client-Server Assignment for Internet Distributed Systems

International Journal of Scientific Engineering and Technology Research
Volume.03, IssueNo.29, October-2014, Pages: 5910-5915

TABLE 1: OPTIMALITY OF $F$: $\frac{F - F_{\text{worst}}}{F_{\text{best}} - F_{\text{worst}}}$

<table>
<thead>
<tr>
<th>$M = 1, \beta = 0.3$</th>
<th>RCD</th>
<th>BCO</th>
<th>NC</th>
<th>Graclus</th>
<th>RND</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\beta = 0.5$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power Law</td>
<td>94.6%</td>
<td>94.6%</td>
<td>94.6%</td>
<td>94.6%</td>
<td>94.6%</td>
</tr>
<tr>
<td>Random</td>
<td>94.6%</td>
<td>94.6%</td>
<td>94.6%</td>
<td>94.6%</td>
<td>94.6%</td>
</tr>
<tr>
<td>Regular</td>
<td>94.6%</td>
<td>94.6%</td>
<td>94.6%</td>
<td>94.6%</td>
<td>94.6%</td>
</tr>
<tr>
<td>$\beta = 0.6$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power Law</td>
<td>94.6%</td>
<td>94.6%</td>
<td>94.6%</td>
<td>94.6%</td>
<td>94.6%</td>
</tr>
<tr>
<td>Random</td>
<td>94.6%</td>
<td>94.6%</td>
<td>94.6%</td>
<td>94.6%</td>
<td>94.6%</td>
</tr>
<tr>
<td>Regular</td>
<td>94.6%</td>
<td>94.6%</td>
<td>94.6%</td>
<td>94.6%</td>
<td>94.6%</td>
</tr>
</tbody>
</table>

B. Optimality

To verify how close the outputs by our algorithms are to the optimal solutions, we made the following examination:

1. For each graph, we calculate $F$ exhaustively for all MN combinations of $X$. Herein, we suppose the best (smallest) $F = F_{\text{best}}$ and the worst (largest) $F = F_{\text{worst}}$.

2. For each graph, calculate $F$ by each of the BCO-V, BCO-E, NC, Graclus, and a randomly generated $X$ (RND), and then calculate $F$’s optimality.

We also obtain each $F$’s ranking ($F_{R}$) out of $M \times N$ outputs, and then calculate its optimality $1 - \frac{F_{R} - 1}{M^{N} - 1}$. The larger those values are the better optimality we have.

3. Do 1 and 2 for
   - A hundred random graphs generated by Barabasi-Albert power-law graph generator algorithm.
   - A hundred different graphs generated by our random graph generator. In our random graph generator, each vertex is allocated at most 10 randomly selected neighbors.
   - Regular graphs in which every node has the equal number of neighbors ($H$) with an equal edge weight. Note $H$ is even and vertex $i$ is connected to $i + 1, \ldots, i + \frac{H}{2}, i - 1, \ldots, i - \frac{H}{2}$. We simulated for $H = 2, 4, \ldots, N/2$ that is, $\frac{N}{2}$ different regular graphs for a given $\{M, N\}$.

4. Do 1 and 3 for $\{M, N\} = \{2, 30\}, \{3, 20\}$.

Tables 1 and 2 show average $\frac{F - F_{\text{worst}}}{F_{\text{best}} - F_{\text{worst}}}$ and $1 - \frac{F_{R} - 1}{M^{N} - 1}$ values for a hundred graphs created by each of the power-law, random and regular graph generators, respectively. Overall, the BCOs for $\beta = 0.3$ or 0.1 show the best optimality in spite of the simple and cheap quantization technique. In fact, unlike the NC, the BCOs constantly output good $F$ values (close or equal to $F_{\text{best}}$) regardless of the graph type. The simulation shows that the BCOs are very suitable for solving our problem. On the other hand, the NC tends to isolate vertices that do not have strong connection to others. This is typically observed in power-law graphs for $M = 2, N = 30$. As a result, though we have small amount of inter server communication $F_c$, the load balance metric $F_l$ becomes large and $F$ produced by NC is larger (worse) than those produced by our algorithms. The Graclus exhibits more balanced cuts than the NC for any type of graph. This is due to its multilevel process; balancing the size of each group is somehow related to balancing the weights of associated edges in small graphs. As a result, the Graclus achieves smaller $F_l$ and $F$ than the NC. However, it does not necessarily result in balancing the total weight of associated edges of each group, and therefore the Graclus does not perform better than the BCOs. As for the computation time, all the methods finish clustering a graph literally in a moment ($\leq 1$ second).

TABLE II: Optimality of $F_{R}$ ($F'$'s Ranking): $1 - \frac{F_{R} - 1}{M^{N} - 1}$

C. Experiments for Larger Power-Law Graphs

As described in Section 1.1, our algorithms should perform better than the NC and Graclus for power-law graphs. We simulated for a hundred power-law graphs, in which each vertex is connected to up to a hundred neighbors, with $M = 4, 7, 10$ and $N = 1,000$. In this setting, we cannot find the rankings for each algorithm as it requires an exhaustive search over all possible assignments which are infeasible for large $N$. Instead, Table 3 shows the average $F$, $F_c$, $F_l$ values...
and \( \frac{l_{\text{max}}}{l_{\text{min}}} \), where \( l_{\text{max}} \) and \( l_{\text{min}} \) are the maximum and minimum elements i.e., the maximum and minimum communication load in \( M \) servers, respectively. \( \frac{l_{\text{max}}}{l_{\text{min}}} \) is another side metric that reflects the load balance among the servers. As shown by the \( F_i \) and \( \frac{l_{\text{max}}}{l_{\text{min}}} \) values, the BCO-V and BCOE fairly balance the load, and at the same time maintain low total communication load as seen in their \( F_i \). Though the NC yields low \( F_i \), it does not balance the load, which appears as large \( F_i \). \( F \) and \( \frac{l_{\text{max}}}{l_{\text{min}}} \) values consequently. The Graclus performs more balanced cuts than the NC, but its \( \frac{l_{\text{max}}}{l_{\text{min}}} \) values are higher than 2, which will not be acceptable in real distributed systems. Interestingly, the \( F_i \) values of the Graclus are very close to those of the BCOs. Hence, the differences in their \( F \) values are mainly determined by their \( F_i \) values; \( F_i \)s of the Graclus are higher than those of the BCOs. This also substantiates that our algorithms aptly strikes the balance between the two opposing metrics: reducing the total communication load and load balance.

Also, the BCOs reduce the total communication load (= 1+ \( F_i \)) by 33-36 percent compared to the random assignment. This indicates that a system that uses 100 servers with a random client-server assignment requires only 64-67 servers (or less because the inter server communication will also decrease by reducing the number of servers) with an assignment by the BCOs. Also, since 1 \( \leq 1+ F_i \leq 2 \) the maximum reduction rate of the total communication load = 50 \%( =1/2). Thus, the reduction rates of 33-36 percent are significantly high, and we can also know how inefficient the random assignment is, though it yields “not bad” load balance (see their \( \frac{l_{\text{max}}}{l_{\text{min}}} \) values). This also verifies the effectiveness of our algorithms. As for the computation time taken for clustering a graph, the Graclus and NC finish within a second, while the BCOs take 18-58 minutes. The BCOs for \( N = 7 \) take longer than those for \( N = 10 \). This is because as described in the end of, we examine binary split twice for odd \( M \), and therefore the initial split for \( M = 7 \) takes longer than that for \( M = 10 \). We recognize the expensiveness of our algorithms. The improvement in the time complexity is the next step of our research.

V. CONCLUSION

In this paper, we present a mathematical model and an algorithmic solution to the client-server assignment problem for optimizing the performance of a class of distributed systems over the Internet. We show that in general, finding the optimal client-server assignment for some pre-specified requirements on total load and load balancing is NP-hard, and propose a heuristic via relaxed convex optimization for finding the approximate solution to the client-server assignment problem. Our simulation results indicate that the proposed algorithm almost always finds the optimal solution. Furthermore, the proposed algorithm outperforms other heuristics, including the popular Normalized Cuts algorithm.

VI. REFERENCES


