Optimized Software Component Allocation
On Clustered Application Servers

by
Hsiauh-Tsyr Clara Chang, B.B., M.S., M.S.

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Professional Studies
in Computing

at

School of Computer Science and Information Systems

Pace University

April 2004

We hereby certify that this dissertation, submitted by Hsiauh-Tsyr Clara Chang, satisfies the dissertation requirements for the degree of *Doctor of Professional Studies in Computing* and has been approved.

_____-_____

Lixin Tao Date
Chairperson of Dissertation Committee

_____-_____

Fred Grossman Date
Dissertation Committee Member

_____-_____

Michael Gargano Date
Dissertation Committee Member

School of Computer Science and Information Systems
Pace University 2004

Abstract

Optimized Software Component Allocation On Clustered Application Servers

by
Hsiauh-Tsyr Clara Chang, B.B., M.S., M.S.

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Professional Studies
in Computing

April 2004

In the last decade, online e-commerce businesses, represented by the e-commerce portals, have grown significantly and become an important sector of world economy. This dissertation helps address the server scalability problem for supporting the sustainable growth of the online e-commerce industries.

Most of today's e-commerce portals are implemented with distributed component technologies and server clusters. Each server application comprises dozens or hundreds of distributed software components, and each of such components can run on any of a cluster of application servers connected by a high-speed fiber local area network (LAN). While multiple server machines support parallel execution of the software components, inter-server communication is a few orders slower than servers' CPU speed. This research studies the optimized allocation of software components to server machines to maximize computation load balance and minimize communication overhead.

Multi-way graph partitioning is first adopted to model the software component allocation problem. The problem is proved to be NP-hard. A novel graph transformation is introduced to combine the two conflicting objectives into a single objective function, and a transformation theorem is proved that problem instances before and after this transformation are equivalent. Based on careful observation of the properties of the solution space, a scheme for incremental objective function evaluation is designed to speed up any iterative solution heuristics to this problem by a factor proportional to the number of software components involved. Simulated annealing is adopted to solve the problem. Extensive experimental study shows that the proposed simulated annealing algorithm can outperform repeated random running in the same amount of time by 20% to 2366.67%, and outperform local optimization by 1.96% to 1000% with a running time about 6 to 100 times of that for the latter.

The major contributions of this research include using multi-way graph partitioning to model a challenging performance problem critical to sustainable growth of e-commerce portals, creative problem transformation for simplifying a complex problem, and incremental objective function evaluation that can benefit any iterative solution heuristics.

Acknowledgements

Earning a doctorate degree has been a difficult journey that was made easier by the support of several important people in my life. This dissertation is the result of that journey, and I am pleased to have the opportunity to express my gratitude to them.

The first person I would like to thank is my direct supervisor, Dr. Lixin Tao. He encouraged me to set high standards for my work, and his enthusiasm for my research will leave a lasting impression on me. Completing this dissertation was made possible because of Dr. Tao's advice, and I own him a great debt of gratitude for all he taught me. Besides being an excellent supervisor, I will always cherish the friendship that I've developed with Lixin.

I would also like to thank my other committee members, Dr. Fred Grossman and Dr. Michael Gargano, who monitored my work and provided valuable feedback on the dissertation.

Finally, I would like to express my gratitude to my mom and dad, Li-Hui and Tao-Hui, my aunt and uncle, Lucy and David, and my sister Teresa. Their encouragement and support helped me during the most difficult times of the doctoral program. Also, I especially want to thank my best friend Cheng-Kai, whose love and support enabled me to complete this work.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

In the last decade, online e-commerce businesses, represented by the e-commerce portals, have grown significantly and become an important sector of world economy. This dissertation helps address the server scalability problem for supporting the sustainable growth of the online e-commerce industries.

1.1 Distributed Software Components as a New Trend of IT Industries

Since early 1990s, the IT industries have been shifting their design and implementation technologies to software frameworks and architectures based on distributed software components [17][20] to better control the software complexity, promote specialized computing and system integration, and support middleware for object-oriented networking. A *software component* is a software unit that usually exists in binary form, exposes a public *Application Programming Interface (API)*, and is independently deployable [20][16][10][18]. Example software component technologies include Microsoft's dynamic linking libraries, Active-X controls, and COM components and Java's JavaBeans. Software component approach completely separates the usage and implementation of a software component, and makes it sharable by multiple applications and easily replaceable. A *distributed software component* technology further supports communication abstraction, and uses a generic software framework to provide

transparent communication ability to network-blind software components. Example distributed software component technologies include Microsoft's DCOM and COM+ [18], Java's Enterprise JavaBeans (EJB) [16], and Object Management Group's CORBA components [10]. Since 1995, the US Department of Defense mandated that all of its contracted software projects must be implemented with software component technologies [20].

A related new development of the last decade is the ubiquitous networking. Web and Internet technologies have made online e-business an important branch of world economy. A typical e-commerce portal has three tiers: the presentation tier (running on a Web server) for generating presentation documents for a client's Web browser to render, the business logic tier (running on an application server) for implementation of business logics for the portal, and the data tier (supported by databases). Both the Web servers and the application severs are based on distributed component technologies. For example, both Java's servlets running in a Java servlet container and Microsoft's ASP pages running on an IIS Web server are (converted into) distributed software components (in a more general sense), so are the EJBs running in an EJB container on an application server or the COM+ components running on a .NET Transaction Server [20].

A major challenge for today’s e-commerce portals is their scalability: whether a portal can provide fast response when the number of their concurrent clients increases. To provide such scalability, a heavy-duty portal, like yahoo.com, typically uses a cluster of dozens of server machines, connected through a (relatively) fast fiber local area network (LAN), for both of its Web servers and application servers. Since the distributed software components can be independently deployed in any server containers on any of the server

machines and communicate with each other transparently, they can take advantage of the hardware parallelism among the server machines to improve the portal scalability.

For a particular hosted computation, it will not be over until all of its employed components finish. Each software component may have different computation load for a particular use case. Each server machine also may have its own particular computing ability based on its resources like CPU speed and memory size. While a fiber-based LAN is faster than its copper version, sending a message through it is still a few orders slower than today's CPU speed (partially due to software overhead for buffer copying to support layered implementation). Communications between two components are much slower if they are assigned to two different server machines than to the same one. Now we have the basic form of software component allocation problem: for a particular computation, what is the optimal allocation of the participating software components to the server machines so that the computation workload of all the involved server machines are balanced, and the total load of communications between any pair of involved components is minimized. We can notice that there are here two conflicting objectives. This problem will be more complicated when we also consider the management of client session data, or when different stages of a computation may have different computation and communication patterns, as we will see in the next section.

While the World Wide Web has had big impact on our society, it only supports a limited form of client-server software architecture. The IT industries have started to work on the next wave of Internet revolution: the Application Service Provider model of computing [20], by which software applications will be maintained by domain experts on service-provider servers and accessed by clients with Web browsers through service provider's

portals. This paradigm eliminates software installation on client's computers, and promotes specialized computing and service integration. The success of this new paradigm also heavily depends on whether we can run hosted applications on clustered servers efficiently. Therefore the importance of the study of efficient software component allocation problems goes beyond today's Web servers and application servers.

1.2 Software Component Allocation Problems

The potentially important software component allocation problems can be divided into two categories: *static* and *dynamic*, depending on whether the optimized allocations can be computed off-line or whether the components can migrate across server machines during execution. Multiple processors could be tightly coupled inside a single server machine; and a subset of the software components may need to maintain its unique session data to serve a particular remote client. All these make software component allocation problems different from the traditional job scheduling on distributed systems [1][21].

A server cluster typically comprises dozens (50 or more) of server machines, each of which may have different computation speed, connected by a fiber LAN. In this study we limit our attention to bus-type LAN, the dominant one in today's IT industries, for which all messages will share the same LAN bandwidth, and at any instance, there could be at most one sender but multiple receivers. The speed for a message to travel the LAN is much lower than the CPU speed of the server machines.

A hosted software application is typically made up of dozens to hundreds of software components that could be distributed in any of the component containers running on the

server machines. Without loss of generality, we assume each server machine will run one component container. For each typical hosted computation, each of the involved software components has an average computation load and an average communication load with each of the other participating components. Since the hosted applications are designed for providing well-defined set of specialized services, it is reasonable to assume that these average computation load and communication load values could be obtained by profiling the applications on a single server machine (similar to Unix profiler utility *prof*).

Now we can model, simplified for essence, a software component allocation problem as a multi-way graph partitioning problem. We abstract a hosted application as an undirected graph $G = (V, E)$ in which V is set of vertices and E is set of edges, each vertex represents a software component and each edge represents a runtime communication requirement. Let function $w_1 : V \to \Re^+$ ($\Re^+$ is the set of positive real numbers) represent the average computation load of the software components, and function $w_2 : E \to \Re^+$ represent the average communication load of the communication requirements. Assume the software components need to run on m server machines, and $\pi : V \to \{1, 2, \cdots, m\}$ represents one component assignment. For each $1 \le i \le m$, we use $P_\pi(i)$ to denote the partition of the vertices (software components) assigned by π to server machine i; a fixed real rate r_i to represent the relative computing ability of server machine i (a larger rate implies a slower execution); and $w_1(P_\pi(i)) = \sum_{v \in P_\pi(i)} w_1(v)$ to represent the total computation load of components assigned to partition i. Let the importance of computation time on the server machines relative to the communication time on the LAN

be represented by a real ratio $0 < t < 1$. Now the question is, how to find an optimal assignment $\pi : V \to \{1,2,\cdots,m\}$ to minimize the objective function

$$f(\pi) = t \cdot W_1(\pi) + (1-t) \cdot W_2(\pi)$$

where $W_1(\pi)$ represents the degree of load balance as defined below

$$W_1(\pi) = \sum_{1 \le i < j \le m} | r_i \cdot w_1(P_\pi(i)) - r_j \cdot w_1(P_\pi(j)) |$$

and $W_2(\pi)$ represents the total communication cost as defined below

$$W_2(\pi) = \sum_{\substack{e=\{u,v\} \in E \\ \pi(u) \neq \pi(v)}} w_2(e)$$

Here the summation operator reflects our assumption that all communications share the same LAN bandwidth.

Since the basic graph bisection problem, for which the partition number is two and the vertices and edges have uniform weights, is NP-complete [1] and a special case of our simplified formulation, all the software component allocation problems described in this proposal are NP-hard.

1.3 Methodologies

The component allocation problem has many variations based on different assumptions, and this research will focus on the one where the communication cost needs to be minimized under the constraint that the computation workload is evenly distributed.

Mathematical modeling is the foundation of this research. The properties of the mathematical model will be studied to derive a problem transformation algorithm that can convert the two-objective-function optimization problem into an equivalent one with a single objective function. For efficient problem solution, solution space neighborhood will be designed to support incremental evaluation of the objective function, which can benefit any solution algorithm based on iterative solution searches. Simulated annealing is chosen as the meta-heuristic for deriving a solution heuristic because it has an explicit strategy to escape from local minima. Experimental comparisons will be conducted between the proposed simulated annealing algorithm and repeated random solutions generated in the same amount of time, and between the proposed simulated annealing algorithm and local optimization for both solution quality and running time.

1.4 Major Contributions

The major contributions of this research include:

- Using multi-way graph partitioning to model an important application server performance problem critical to the sustainable growth of online e-commerce industries.

- Proving that this problem is NP-hard, so no efficient algorithms could ever be designed to produce optimal solutions to it in practical time.

- Designing a problem transformation algorithm to convert the problem with multiple objective functions into an equivalent typical combinatorial optimization problem with a single objective function.

- Designing a scheme for incremental objective function evaluation that can improve the performance of any iterative solution heuristics.
- Deriving an efficient heuristic solution based on simulated annealing, and studying the sensitivity of the heuristic to its various parameters.
- Designing experiments to study the performance of our heuristic relative to repeated random solutions and local optimization.

1.5 Dissertation Outline

The dissertation consists of six chapters described in the following manner:

Chapter 1 introduces the important scalability problem of E-commerce portal servers and the associated software component allocation problem, presents the solution methodologies and major contributions of this research.

Chapter 2 provides surveys of commonly used meta-heuristics for combinatorial optimization problems and describes the characteristics of each heuristic.

Chapter 3 describes the problem formulation for multi-way graph partition and problem transformation.

Chapter 4 provides the design of solution space neighborhood as well as the incremental evaluation of the objective function.

Chapter 5 provides the design of a simulated annealing heuristic for the proposed software component allocation problem, and conducts sensitivity analysis to its various parameters and cooling schedule.

Chapter 6 uses extensive experimental comparisons to study the performance of the simulated annealing algorithm relative to repeated random solutions and local optimization.

Chapter 7 concludes with some observations and future work.

Chapter 2

Graph Partitioning and Solution Heuristics

This chapter surveys the graph partitioning problems as well as the major meta-heuristics for combinatorial optimization.

2.1 Graph Partitioning

Graph partitioning is one of the richest fields of computing algorithms, with wide applications in parallel processing, distributed computing, VLSI design and layout, network partitioning, distributed database design, and sparse matrix factorization [4][12][1][5][22]. The most popular heuristics for graph partitioning include the Kernighan-Lin algorithm (KL) [13] for graph bisection and its enhancement variation [4]. Johnson et al. [12] performed an extensive study of the simulated annealing algorithm for the graph bisection problem and observed that simulated annealing on the average performed better than KL. Bui et al. [1] developed a genetic algorithm for multi-way graph partitioning, and conducted extensive experimental evaluations of the related algorithms to show its superior performance. Tao et al. [21], as well as many other researchers, used graph partitioning to address the problem of optimized allocation of processes/jobs to the processors in a distributed environment. Tao et al. [22] proposed *stochastic probe*, a new effective and generic meta-heuristic, and demonstrated its superior performance in multi-way graph partitioning.

The existing studies [12] of graph partitioning usually simplify the problem constraints described in this research by dropping the weights of the vertices or edges.

2.2 Solution Heuristics

For NP-hard problems, we can only obtain optimal solutions for small problem instances. For practical problem instance sizes, heuristics must be used to find optimized solutions within reasonable time frame. Unlike algorithms, heuristics do not guarantee to generate optimal solutions. A *heuristic* is an algorithm that tries to find good solutions to a problem but it cannot guarantee its success. Most heuristics are not established on rigid mathematical analysis, but on human intuitions, understanding of the properties of the problem at hand, and experiments. The value of a heuristic must be based on performance comparisons among competing heuristics. The most important performance metrics are solution quality and running time.

The term *meta-heuristics*, first introduced in Glover [6], derives from the composition of two Greek words. The suffix *meta* means "beyond, in an upper level" and *heuristic* means "to find, discover". A *meta-heuristic* is a strategy that guides the search process, or an abstraction of a class of similar heuristics. Meta-heuristics are approximate and usually non-deterministic, not problem-specific. They may incorporate mechanisms to avoid getting trapped in confined areas of the search space. The basic concepts of meta-heuristic permit an abstract level description, and may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy. More advanced meta-heuristics are used to guide the solution searches today [1]. To effectively

resolve a problem based on a meta-heuristic, we need to have more understanding of the characteristics of the problem, and creatively design and implement the major components of the meta-heuristics. As a consequence, using a meta-heuristic to propose an effective heuristic to solve an NP-hard problem is an action of research.

In the following we outline the most important meta-heuristics from a conceptual point of view.

2.2.1 Local Optimization

A general heuristic search technique, *local optimization* is also called *greedy algorithm* or *hill-climbing*. It attempts to improve on the solution by a series of incremental, local changes. Each move is only performed if the resulting solution is better than the current solution. The algorithm stops as soon as it finds a local minimum. The high level algorithm is sketched in Algorithm 1.

Algorithm 1 Local optimization

1. Get an initial solution S.
2. While there is an untested neighbor of S do the following.
   2.1 Let S' be an untested neighbor of S.
   2.2 If $cost(S') < cost(S)$, set $S = S'$.
3. Return S.

Local optimization starts from a random initial solution, and it keeps migrating to better neighbors in the solution space. If all neighbors of the current partition are worse, then the algorithm stops. This scheme can only find local optimal solutions that are better than all of their neighbors but they may not be the global optimal solutions, as illustrated in Figure 1.

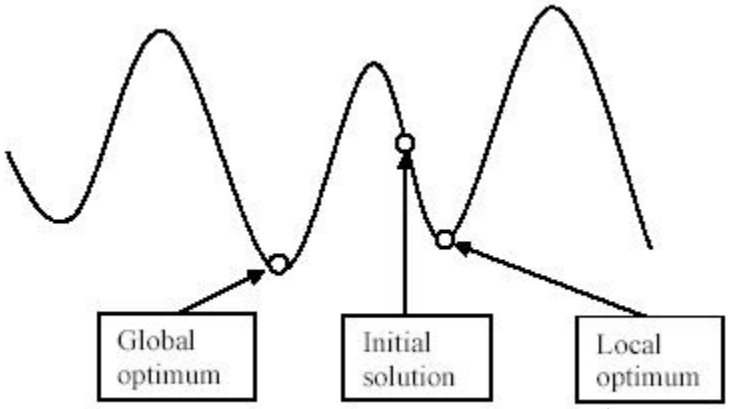

Figure 1 Local vs. global solutions

2.2.2 Genetic Algorithm

Genetic algorithm [4] is an iterative procedure maintaining population of structures that are candidate solutions to specific domain challenges. During each generation the structures in the current population are rated for their effectiveness as solutions, and on the basis of these evaluations, a new population of candidate structures is formed using specific "genetic operators" such as reproduction, crossover, and mutation. It is based on the analogy of combinatorial optimization to the mechanics of natural selection and natural genetics. Its application in combinatorial optimization area can be traced back in early 1960s [8].

A genetic algorithm starts with a set of initial solutions (chromosomes), called a population. This population then evolves into different populations through a series of iterations. At the end, the algorithm returns the best member of the population as the solution to the problem. For each iteration or generation, the evolution process proceeds as follows. Two members of the population are chosen based on some probability

distribution. These two members are then combined through a crossover operator to produce an offspring. With a low probability, this offspring is then modified by a mutation operator to introduce unexplored search space to the population, enhancing the diversity of the population (the degree of difference among chromosomes in the population). The offspring is tested to see if it is suitable for the population. If it is, a replacement scheme is used to select a member of the population and replace it with the new offspring. Now we have a new population and the evolution process is repeated until certain condition is met, for example, after a fixed number of generations. This genetic algorithm generates only one offspring per generation. Such a genetic algorithm is called steady-state genetic algorithm [24][19], as opposed to a generational genetic algorithm that replaces the whole population or a large subset of the population per generation. A typical structure of a steady-state genetic algorithm is given in Algorithm 2 [2].

Algorithm 2 Genetic algorithm

1. Create initial population of fixed size.
2. Do the following
   2.1 Choose parent1 and parent2 from population.
   2.2 Offspring = crossover (parent1, parent2).
   2.3 Mutation (offspring).
   2.4 If suited (offspring), then
       Replace (population, offspring);
   Until (stopping condition).
3. Return the best answer.

2.2.3 Simulated Annealing

Simulated annealing is commonly [4][12] said to be the oldest meta-heuristic that has an explicit strategy to escape from local minima. The origins of the algorithm are in statistical mechanics (Metropolis algorithm) and it was first presented as a search