Jan Stender, Silvan Kaiser, Sahin Albayrak

Mobility-based Runtime Load Balancing in Multi-Agent Systems

Zuerst erschienen in:

18th International Conference on Software Engineering and Knowledge Engineering (SEKE'06), Reedwood City, CA, USA

Mobility-based Runtime Load Balancing in Multi-Agent Systems

Jan Stender[1], **Silvan Kaiser**[1], **Sahin Albayrak**[1]

[1]DAI-Labor, Technische Universität Berlin, Secretary GOR 1-1,
Franklinstrasse 28/29, D-10587 Berlin, Germany
{jan.stender, silvan.kaiser, sahin.albayrak}@dai-labor.de

Abstract

Supported by the development of Grid Computing technologies, dynamic resource sharing is becoming an icreasingly important issue in the area of distributed computing. In this paper, an infrastructure for dynamic runtime load balancing based on mobile agents is introduced. The approach involves a migration of active agents between hosts in order to exploit distributed system resources to the best possible extent. Service provisioning agents are continuously reachable and overloaded hosts can relocate the agents to other, less loaded hosts at any time. Experimental results show that the approach is suitable for small-scale systems.

Keywords: Load Balancing, Mobile Agents, Grid Computing

1. Introduction

The inherently distributed nature of Multi-Agent Systems brings about a strong relation to the world of Grid Computing. The broad range of common attributes is significant and is stressed in the Grid domain as well as in the domain of software agents [4, 6]. Inspired by the achievements of present day Grid computing technology, Multi-Agent Systems can benefit from the development of uniform, secure and reliable mechanisms for dynamic resource sharing. Grids use load balancing mechanisms in order to allow multiple resource consumers a simultaneous allocation of distributed computing resources. In this paper, an approach for transferring this concept to the agent world is introduced.

Since autonomous agents may consume resources in an ever-changing and barely predictable manner, Grid resource allocation mechanisms do not meet the requirements of agent communities, as they generally schedule the allocation of resources at job startup time rather than runtime. Therefore more dynamic and adaptive load balancing strategies are needed which allow a fast response to changes in the resource consumption behavior of single agents. A possible solution to this problem is a flexible runtime load balancing that utilizes mobile agent technology. The concept relies on a community of mobile agents that dynamically arrange their physical locations, such that available resources are exploited to the best possible extent. Design and implementation of a load balancing infrastructure which involves a reasonable trade-off between migration overhead and quality of load distribution will be described in the following. Experimental results show the viability of the approach.

2. Problem Description

In general, Multi-Agent Systems have a distributed character. Most agent-based applications are designed in a way that agents are spread across multiple hosts. Once started, such agents usually do not change their physical locations during their lifetime.

While agents are busy, they consume system resources, such as CPU capacity, memory or communication bandwidth. If an agent is permanently bound to a certain host, it always depends on the resources made available by that host. Thus the quality of the services provided by the agent may suffer if the host runs out of resources. Moreover, spare resources in the system might not be used in an optimal manner, as it is possible that a different host is more appropriate to run the agent. A fixed arrangement of agent locations therefore makes it hardly possible to guarantee that resources are optimally used throughout the system. Since agents react to events, e.g. service provisioning requests, their activity potentially occurs asynchronously and is nearly unpredictable. Thus, agent-based applications often underlie an ever-changing demand for resources on all machines.

In order to solve this problem, a self-organized load balancing infrastructure is needed which can quickly respond to changes in the demand for system resources.

3. Agent Mobility and Load Balancing

Mobility is a feature of various Multi-Agent Systems which cuts into weak and strong mobility. While a weakly mobile agent loses its execution state when migrating, a strongly mobile agent is capable of seamlessly continuing its execution at the destination host. Strong agent mobility generally involves three steps: an interruption of the agent's activity on the local host, a transfer of the agent including its program code and state to the remote host and a resumption of the agent's activity at the remote host. The majority of mobility-enabled agent systems are restricted to weak mobility. There are a few ones supporting strong mobility, including JIAC [5], NOMADS [8] and Organic Grid [2]. With the aim of implementing strong agent mobility, NOMADS is built on top of a special Java Virtual Machine which is capable of capturing and restoring the execution state of a Java thread on different machines. Organic Grid relies on a preprocessor that generates Java code from code written in an enhanced Java-based language. The approach adopted by JIAC is to provide a custom language for the implementation of agents which is interpreted at runtime. The execution states of all JIAC agents are part of the JIAC runtime system and can hence be restored on any JIAC-enabled machine.

Strong agent mobility combined with platform independence set the stage for a dynamic resource sharing infrastructure, as provided by various Grid systems. However, most Grid computing infrastructures are designed to only decide at which host to start a job and do not redistribute load in a dynamic manner. There are some systems such as Condor [10] that support load balancing at runtime based on a checkpointing mechanism. When load is getting too high on a single host, jobs running on it can be checkpointed and moved to a different host, i.e. the entire job execution state is captured on the local host and restored on the remote host. This, however, requires a totally homogeneous computing environment, whereas JIAC is Java-based and hence can be run on nearly any machine.

There are some alternative approaches to the problem of Grid load balancing with mobile agents, such as implemented by the Messor system [7]. Messor is capable of arranging a uniform distribution of jobs across peers in large-scale peer-to-peer systems. Agents carrying the jobs migrate between peers, implementing the emergent behavior of an artificial ant colony. However, the job redistribution mechanism is restricted to jobs that are queued, i.e. it does not involve running jobs. This also applies to the Organic Grid [2] approach. Organic Grid manages large tasks with the aid of mobile agents that are cloned on different hosts. Each clone executes a different range of subtasks of the original task, thereby spreading system load across the different nodes. Subtasks that have been started, however, cannot be relocated to a different host. Organic Grid hence also relies on startup time scheduling.

The JIAC infrastructure follows a service based approach. Agents provide services that offer specific functionalities. Service provisioning agents can be either inactive while waiting for new requests or be very busy, using up resources while reacting to a service request at other times. As it is hardly perceivable which host in a Multi-Agent System will be busy at a given time, a runtime balancing mechanism is needed. This allows load distribution in an environment that prohibits load forecasting and therefore renders scheduling at startup time almost useless.

In service-oriented computing infrastructures, downtimes of service providers ought to be reduced to the minimum. Since JIAC offers relocation transparency, service requests never get lost, not even while the service provisioning agent is migrating. As soon as the agent activity is resumed on the target host, service provisioning activity is resumed as well. As opposed to this, systems like *While You're Away* [9] involve a queueing of agents if the network does not provide sufficient resources which causes these agents to be inoperable until execution is continued. Moreover, JIAC does not assume that agents are independent of each other. JIAC agents may communicate in a location transparent fashion and may hence invoke services provided by any other agent at any time.

A placement of agent replicas on different hosts combined with a load-dependent provider selection for each service request might also be a viable approach to a dynamic load balancing system. However, such an approach lacks flexibility in connection with resource-intensive service provisioning activity. It brings about similar problems like the aforementioned Grid systems, in that tasks are scheduled before being executed and cannot be redistributed at runtime.

4. Approach

Dynamic load balancing is a possible application for Agent mobility [1]. In a network of hosts at which agents are running, an agent suffering from lack of free resources can simply be transferred to a remote host where a sufficient amount of resources is available. If such migrations take place in a coordinated fashion, load can be dynamically distributed in way that competition for resources between agents is significantly reduced.

Systems like Comet [3] and WYA [9] have delivered a proof of concept for mobility-based agent load balancing. It has turned out that the ability of agents to change their physical locations in a computer network can be efficiently utilized for load distribution purposes. The kind of agents which systems like Comet and WYA are based on, can be compared to mobile processes, whereas JIAC agents rather have the character of autonomous service providers.

Since services provided by JIAC agents usually remain unused for the most time, it is at best difficult to foresee the behavior of such agents with regard to resource consumption. Due to the asynchronous character of cooperative Multi-Agent Systems, JIAC agents can consume system resources in an ever-changing and barely predictable manner. A load balancing infrastructure for JIAC agents therefore has to quickly respond to changes in the demand for resources of single agents, in order to guarantee full functional capability of the system at any time.

This raises some general questions which have to be taken into consideration with regard to the design of a mobility-based load balancing infrastructure:

Under which circumstances should migrations be initiated? In general, it is important to trigger migrations not too frequently, as a migration implicates some computational and networking overhead, as well as a delay of the agent's activity. On the other hand, the system ought to be capable of quickly responding to significant changes in the demand for resources.

The decision whether and when a migration is initiated should be made locally rather than globally, in order to reduce the risk of creating single points of failure and resource bottlenecks in large systems consisting of many hosts. In general, agents should not migrate unless a migration would bring about a more optimal distribution of load throughout the system. Assuming that no agent in the system frees any allocated resources or allocates any free resources, the system finally ought to reach a state in which no migrations are initiated anymore. With respect to this, it is essential to initiate a migration only as a result of a significant change in the demand for resources. Otherwise, oscillation effects might occur, i.e. a ceaseless movement of agents between hosts might be inevitable.

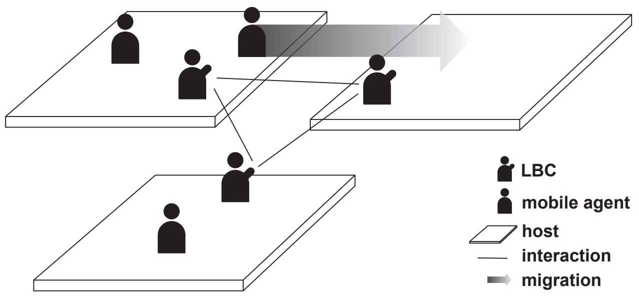

Figure 1. System Architecture

Which agents should be selected for a migration? With regard to this, the main problem is to find out to what extent which local agent is responsible for the load caused on the local host. Unfortunately, this has turned out to be nearly impossible with the JIAC system in connection with certain kinds of system load, such as CPU load. Hence, no satisfactory solution to the problem has been found so far. Agent selection is still a topic for further research.

To which hosts should these agents be moved? It is of little benefit to migrate agents to hosts which do not have a sufficient amount of free resources for additional agents. This would only raise the need for a subsequent migration on the destination host as soon as the agent starts requesting access to the resources provided by that host. Taking into consideration that it is barely possible to assess the exact amount of resources required by a single agent, it is reasonable to assume that hosts with a large surplus of free resources have a higher chance of being able to run an additional agent than others. The lower the utilization of a host is, the more likely it hence should be that the host becomes the destination for the next migration.

5. Implementation

The JIAC load balancing infrastructure requires a dedicated agent on each host to be responsible for the local coordination of load balancing activity. Each such load balancing coordinator (LBC) decides under which circumstances to initiate outgoing as well as to accept incoming migrations. Although LBCs are autonomous, they act as a team in order to establish the load balancing infrastructure. The architecture of the load balancing system is illustrated in figure 1.

Basically, load balancing can be seen as the result of an interaction between LBCs. The fact that the agent discovery service makes it possible for each LBC in the system to find all the other ones, an LBC can find out where to migrate a local agent by asking other LBCs whether their corresponding runtime environments have sufficient resource capacity to host the agent.

With the aim of finding out when a migration has to be initiated, an LBC c continuously collects information about the current level of load l_c on its local host h_c. All LBCs hold load threshold values t_c representing the largest admissible amount of load, i.e. the smallest possible amount of free resources on their hosts. As soon as the threshold is exceeded, i.e. $l_c > t_c$, a rule-based notification mechanism informs the agent that load balancing activity is necessary. Upon receiving such a notification, a check whether to initiate a migration is performed. Two requirements have to be fulfilled:

1. There's a at least one LBC on a remote host with spare resources which is significantly less loaded than the local one. This requirement helps to mitigate the problems of unnecessary migrations and agent oscillation described in the previous section. All LBCs hold a constant value f_c representing the minimum load difference which is required to carry out a migration. At this time, it is still up to the administrator to find reasonable values for f_c.

2. A fixed minimum amount of time has elapsed since the previous successful migration. This time span t is a global system parameter which also has to be set by the administrator. It provides a means to adjust the reactivity of the system. The smaller t is, the more often migrations will take place. Large values for t will allow migrations only every now and then, and hence will give agents more time to perform their tasks at the cost of the speed at which load is balanced. Reasonable values for t depend on the respective application scenarios.

```
proc $balance\_load$($c$: LBC, $\mathcal{R}$: Set of LBC)
begin
  if $(l_c > t_c) \wedge$ ($t$ has elapsed since
      the last migration) then
    $A :=$ {mobile agents on $h_c$}
    for an arbitrary $a \in A$ do
      $P := \{r \in \mathcal{R} \mid l_r \leq min(t_r,\ l_c - f_r)\}$
      $D := \{p \in P \mid \neg\exists p` \in P :\ l_{p`} < l_p\}$
      for an arbitrary $d \in D$ do
        $migrate$($a$, $h_d$)
      done
    done
  endif
end
```

c:	LBC on the local platform	$\mathcal{R}$:	set of all non-local LBCs
l_x:	current load detected by LBC x	t:	global time constant
h_a:	host of agent a	t_x:	load threshold for LBC x
f_x:	minimum migration load difference defined by LBC x		

Figure 2. Load Balancing Algorithm

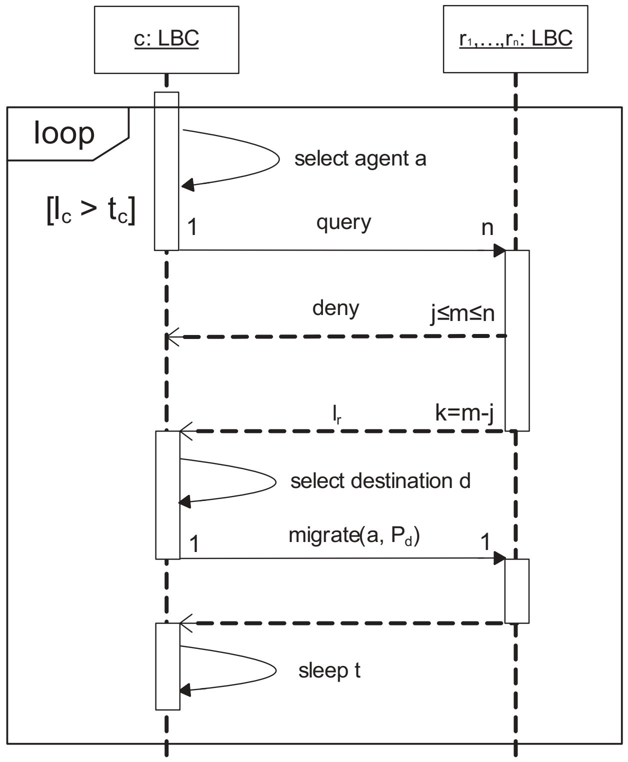

Figure 3. Load Balancing Interaction Protocol

All LBCs use the load balancing algorithm shown in figure 2. In order to provide for a fine-grained redistribution of load in the system, only one agent a is migrated at a time. For the sake of simplicity and due to lack of a good solution to the aforementioned problem concerning agent selection, this agent is chosen by chance. The subset P of potential target LBCs is selected from the set $\mathcal{R} = \{r_1, ..., r_n\}$ of remote LBCs which have runtime environments with enough capacity to host a. The host h_d with the largest amount of free resources eventually serves as the destination host.

Since some of the data used by the algorithm in figure 2 is not locally available for an LBC c, the algorithm is part of an interaction protocol in which all LBCs are involved. Figure 3 illustrates the different steps constituting the interaction. After having selected an arbitrary agent for the migration, a query is sent to all remote LBCs with the intention to find out which remote hosts have spare resources. An LBC r receiving such a query basically does nothing but

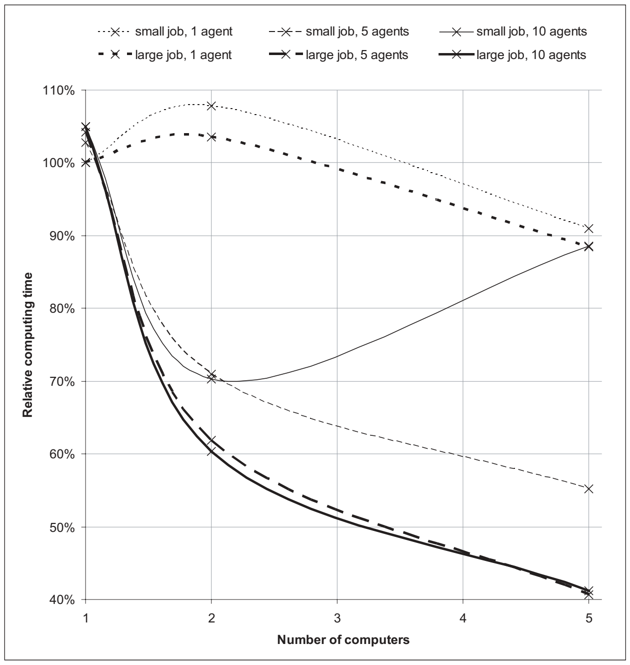

Figure 4. Measurement Results

to check whether its load threshold is exceeded. If so, the reply from r indicates that there is no capacity left for another agent, i.e. accepting the agent would probably cause the load threshold to be exceeded. Otherwise, r replies by sending back its load value l_r which suggests the willingness to accept another agent. A maximum waiting time for replies guarantees that the poll will come to a close. Finally, the least loaded host is chosen and the migration is carried out, followed by a waiting period t.

6. Experimental Results

In order to determine the effectiveness of the load balancing infrastructure, an implementation of a distributed ray tracer has been used. The system consists of a set of ray tracing agents which are responsible for the computation of a picture from an XML-based description of a 3D scene. A dedicated manager agent coordinates the computing activity. When started, it spawns an adjustable number of ray tracing agents. Each of these is assigned the computation of a certain slice of the image. When an agent has completed its task, it initiates a service invocation on the manager agent in order to send back the result. When this has been done by all ray tracing agents, the image is assembled by the manager.

A runtime environment consisting of up to five hosts has been set up, each running on a desktop computer with Windows OS and a CPU clock rate ranging between 600 MHz and 1.2 GHz. Several test series have been carried out, with varying numbers of ray tracing agents and computers. Tests have been repeatedly run in connection with different job sizes. A small job could approximately be completed on a single computer within 10 minutes, a medium job within 30 minutes and a large job within 60 minutes. Load balancing was based on a measurement of CPU load. As for the configuration of the load balancing infrastructure, load thresholds have been adjusted such that all machines can host up to one ray tracing agent without causing the threshold to be exceeded. The ray tracing manager agent and all ray tracing agents were initially started on a certain machine with an average CPU speed.

Figure 4 shows the relative computing time, i.e. the total time needed to compute an image, compared to the time it took on a single desktop computer with an average CPU clock rate. The results reveal that load balancing can significantly reduce the computing time for medium-sized and large jobs. Since the initial distribution of the agents started by the manager takes some time, small jobs could not be distributed in a timely manner. Aside from the forced delay of load balancing activity which is meant to prevent overreactions, this also comes from the fact that a high level of load on a machine brings about a slow-down of the agent management system, which amongst others is needed to carry out migrations. Furthermore, overhead caused by migrations is shown in the ”1 agent” graphs. In the worst case, performance can even be lower than without load balancing. Such a loss in performance originates from random migrations which are a result of shortcomings in the load measurement techniques used.

7. Summary and Outlook

In this paper the concept of a load balancing infrastructure for mobility-enabled Multi-Agent Systems has been introduced. The infrastructure implements a dynamic approach to a runtime distribution of load via agent migrations in a network of hosts. When resources on a host are running short, a certain agent which is responsible for the load balancing coordination tries to find a remote host with enough capacity to take over one of the local agents. For this purpose, load balancing coordinators communicate via a certain load balancing interaction protocol. Experimental results confirm that the approach is suitable for small-scale agent-based applications.

The decentralized approach provides for stability and robustness, due to the fact that there is no single point of failure. Unlike systems such as WYA [9], the JIAC load balancing infrastructure does not interrupt and defer the execution of agents after deploying a fixed number of agents. Instead all agents are running and reachable until system load threatens to block an overloaded system. In this case a fallback mechanism ensures system stability while keeping agent downtime at a minimum.

The load balancing concept offers room for improvements, mainly concerning configuration and scalability. The configuration issue relates to approaches towards an improved automation of the system administration. This involves exploring to what extent it is possible to automatically adjust system parameters, such as load threshold values or migration frequencies. Probabilistic migration decisions might offer an alternative to the deterministic approach.

As far as scalability is concerned, the fact that each LBC communicates with all remote LBCs leads to a high communication overhead in large systems. With respect to this, better alternatives need to be examined, such as hierarchical or decentralized approaches.

As briefly mentioned in section 6, CPU load monitoring is another issue which has to be addressed. Neither a benchmark-based algorithm nor a determination of the CPU load based on data from the operating system kernel deliver reliable load values. In particular, it is hard to find a uniform CPU load monitoring mechanism if different operating systems or multi-processor machines are involved. Mechanisms are needed which enable a transparent monitoring of load caused by single agents, so as to allow for a more efficient selection of agents to migrate.

References

[1] G. Cabri, L. Leonardi, and F. Zambonelli. Weak and strong mobility in mobile agent applications. In *Proceedings of the 2nd International Conference and Exhibition on The Practical Application of Java (PA JAVA 2000)*, Manchester, UK, April 2000.

[2] Arjav Chakravarti, Gerald Baumgartner, and Mario Lauria. The Organic Grid: Self-Organizing Computation on a Peer-to-Peer Network. In *International Conference on Autonomic Computing (ICAC'04)*, pages 96–103, New York, USA, May 2004.

[3] Ka Po Chow, Ricky Y. K. Kwok, Hai Jin, and Kai Hwang. Comet: A communication-efficient load balancing strategy for multi-agent cluster computing. In *Proceedings of Parallel Computing99 (ParCo99)*, Delft, Netherlands, August 1999.

[4] Ian Foster, Nicholas R. Jennings, and Carl Kesselman. Brain meets brawn: Why Grid and agents need each other. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 8–15, New York, New York, 2004. IEEE Computer Society.

[5] Stefan Fricke, Karsten Bsufka, Jan Keiser, Torge Schmidt, Ralf Sesseler, and Sahin Albayrak. Agent-based telematic services and telecom applications. In *Communications of the ACM 44 (4)*, 2001.

[6] M. Luck, P. McBurney, O. Shehory, and S. Willmott. *Agent Technology Roadmap (A Roadmap for Agent Based Computing)*. AgentLink, 2005.

[7] Alberto Montresor, Hein Meling, and Özalp Babaoğlu. Messor: Load-Balancing through a Swarm of Autonomous Agents. In *Proceedings of the International Workshop on Agents and Peer-to-Peer Computing in conjunction with AAMAS 2002*, Bologna, Italy, July 2002.

[8] Niranjan Suri, Jeffrey Bradshaw, Maggie R. Breedy, Paul T. Groth, Gregory A. Hill, and Renia Jeffers. Strong mobility and fine-grained resource control in nomads. In *Proceedings of the Second International Symposium on Agent Systems and Applications and Fourth International Symposium on Mobile Agents, ASA/MA 2000, Zurich, Switzerland*, volume 1882 / 2004 of *Lecture Notes in Computer Science*, pages 2–15. Springer-Verlag, 2000.

[9] Niranjan Suri, Paul T. Groth, and Jeffrey M. Bradshaw. While You're Away: A system for load-balancing and resource sharing based on mobile agents. In *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, pages 470–473. IEEE Computer Society, 2001.

[10] Todd Tannenbaum, Derek Wright, Karen Miller, and Miron Livny. Condor – a distributed job scheduler. In *Beowulf Cluster Computing with Linux*. MIT Press, October 2001.