NETWORK LOAD BALANCING WITH STRONG MIGRATION IN AN AGENT BASED GRID SYSTEM USING CSP APPROACH

Zafer Altuğ Sayar[1] and Prof. Dr. Nadia Erdoğan [2]

[1]Computer and Informatics Faculty, Istanbul Techical University, İstanbul, Turkey
sayar@itu.edu.tr
[2]Computer and Informatics Faculty, Istanbul Techical University, İstanbul, Turkey
nerdogan@itu.edu.tr

ABSTRACT

In this paper we present a dynamic network traffic balancing approach using strong migration on an agent based grid environment. The paper focuses on three different areas, namely load balancing, strong mobility and CSP (Constraint Satisfaction Problem) approach. We use CSP approach for making task migration decisions. Strong task migration is used to move tasks between nodes at runtime to maintain dynamic balancing. Our approach is implemented on an agent based grid system, where all loads can be formulated as weighted constraints of a CSP. We define inter-messaging rates of tasks as the main load of our system. Activated by the results produced by CSP execution, the strong migration mechanism we have integrated into the grid system maintains dynamic traffic balancing by transferring tasks such that frequently communicating tasks end up on the same node and do not consume network bandwidth. Experimental results show remarkable reduction on network use of the grid system.

KEYWORDS

Grid computing, Strong Mobility, Migration, Load Balancing, Csp

1. INTRODUCTION

Grid computing is a popular research topic in computer science. Multi agent systems are recently used in an increasing trend. This paper introduces a dynamic network traffic balancing approach which is implemented on an agent based grid system, an execution environment that combines the advantages of agents and grid computing.

Grids are hardware and software infrastructures which are new approaches of the parallel and distributed computing. Grids support the distribution, sharing and coordinated use of heterogeneous resources [8]. The systems which are connected by a grid might be in the same location or distributed globally, running on different hardware or operating systems, and might be owned by different organizations [7].

Agents are flexible, encapsulated and autonomous software components, which execute an assigned task by interacting with other components in the environment [1]. They are also social components, which are able to communicate and coordinate with other components in the environment. They can collaborate on an execution of a task or make collaboration to achieve a total goal. With the aid of heterogeneity, coordination and distribution power of grids, agent systems become more efficient and adaptive solutions for variety types of tasks.

DOI: 10.5121/ijgca.2012.3404

Performance and efficiency of grids are generally important expectations of users. Grids use effective load balancing algorithms for distribution of tasks to meet these expectations. A load balancing algorithm aims to improve the total utilization of available resources and efficiency of the task assignment in a grid [9]. Load balancing algorithms can be classified into two categories: static or dynamic. In static algorithms, load balancing decision making mechanisms are defined at compile time when resource requirements are estimated. On the other hand a multicomputer with dynamic load balancing allocates/reallocates resources at runtime based on no a priori task information, which may determine when and which tasks can be migrated [4].

Process migration via strong mobility is an efficient mechanism to achieve dynamic load or network traffic balancing. Strong mobility allows the transfer of an executing process from a source site to a distant site, where it resumes the execution from the interruption point. In this work, we used compiled code level strong mobility approach which is implemented in Java. The portability of this technique is independent of operating system and the JVM (Java Virtual Machine).

CSP (Constraint Satisfaction Problem) defines a problem composed of a finite set of variables each of which has a finite domain of values and a finite set of constraints. The Weighted Constraint Satisfaction Problem (WCSP) is a well known soft constraint framework for modeling over-constrained problems with practical applications in domains such as resource allocation, combinatorial auctions and bioinformatics. WCSP is an optimization version of the CSP framework in which constraints are extended by associating costs to tuples [6].

We implemented our load balancing approach on an agent-based grid system (AGrid)[1] that allows for sharing of processing power resources and task executing on remote platforms. We extended the AGrid environment by adding strong mobility and dynamic load balancing features. We used Brakes[2], a byte code transformation library implemented in Java to achieve strong mobility of tasks. We formulated the loads of the grid as weighted CSP constraints. Thus the load balancing decision problem of the grid becomes a weighted CSP solving problem. We used a generic CSP solver library which is implemented in Java, Jacop[10] (Java Constraint Programming solver) to solve the WCSP.

2. RELATED WORK

There are many studies over agent based mobility on grids. MAGDA[5] is a mobile agent based grid architecture which allows weak mobility of agents. NOMADS[13], D'Agents[14] are mobile agent systems that support strong mobility with using specialized JVM. AGrid offers an agent based grid architecture which doesn't focus on mobility.

The mobility based agent environments are mostly focus on weak mobility of agents. We implemented strong mobility of tasks on an agent based grid environment without modifying JVM.

3. PROPOSED AGENT BASED GRID SYSTEM

We used AGrid[1] infrastructure as main grid framework. We implemented a grid system to share processing power and to execute tasks on remote platforms using AGrid. The AGrid system is implemented on JADE[12] (Java Agent Development Framework) which is developed by Telecom Italia SpA. JADE is a distributed runtime environment on which mobile agents can live, communicate and run parallel tasks via behaviours. JADE also supports graphical user interfaces that can be used for debugging, monitoring, logging and management of the agent system. In addition, JADE is compliant with FIPA specifications, which enables the agents to communicate and cooperate with other agent systems which are also compliant with the FIPA standards.

In this section we will introduce AGrid infrastructure. We will mention the components of AGrid and the protocols between agents briefly. We will describe the properties we extended from AGrid, the new features we added to AGrid, migration analysis algorithm and applied strong migration mechanism in detail.

3.1. Agents

In AGrid, four different types of agents cooperate to provide a distributed computing environment.

- **Manager agent** which is in charge of general grid management.
- **Worker agents** which execute jobs assigned to them and produce results.
- **Client agents** which use the grid to run their tasks.
- **Delegate agents** which help the manager agent via coordinating the interaction and the communication between client or worker agents and the manager agent.

The agents above are the base agents of AGrid infrastructure. The detailed information of default implementations of these types of agents is given in [1]. We added new features to existing AGrid agents and created a new type of agent called **Migration Manager agent**.

- **Migration Manager agent** manages migration of tasks in the grid. It observes the interactions between tasks and calculates the possible reducing of network traffic when a task migrates to another worker agent in the grid. After the calculations the migration manager decides the migrations.

"Figure 1" and "Figure 2" shows the grid architecture which is extended with the migration manager agent.

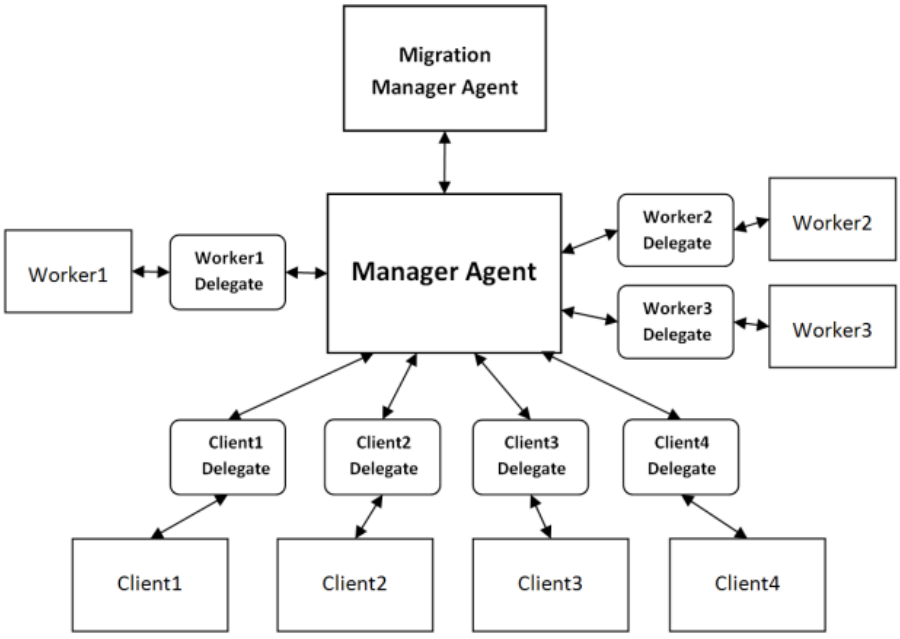

Figure 1. Extended Grid Architecture

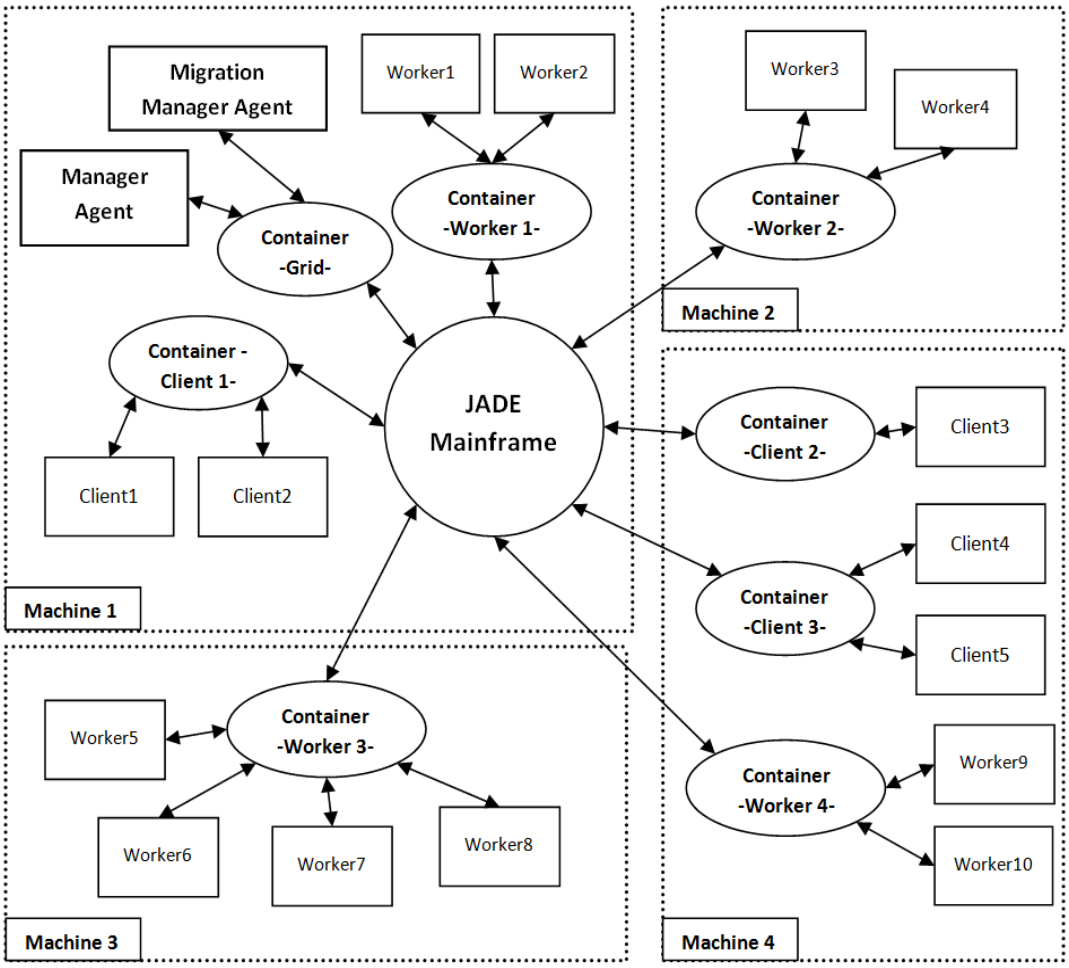

Figure 2. Grid Architecture with JADE Mainframe

In addition to AGrid, tasks implement java.lang.Serializable and java.lang.Runnable to maintain portability. As a result of implementing Runnable interface each task should implement abstract "run" method to do its assigned job. Each task of the system is a runnable for a standard JVM. This allows us to migrate tasks to agents in remote locations of the grid.

3.2. Protocols

Each agent in AGrid system must execute certain protocols during its lifetime in the grid. These protocols define the behaviour of the agent according to the role it carries in the system [1].

Agent connection protocols, task assignment protocol and agent disconnection protocols are the AGRID protocols which we used without changing. The details of these protocols are described in [1]. We have changed some of the protocols, and we also added some new protocols to maintain migrations and portability. This section mentions the protocols which we changed or the protocols are newly added.

3.2.1. Task Running Protocol

This protocol describes task assigning and returning of the execution results. The protocol details are as follows, worker agent accepts the task which is assigned by its delegate agent. Then worker

agent notifies the delegate about the task acceptation. The worker agent requests the agents which it will communicate from its delegate agent, the delegate agent gather the agents sender and receiver agents list from the manager agent and sends the lists to the worker. Then the worker agent creates a task runner object which will create a thread to run the task. The task runner executes the task, after the execution the task runner reports the result to the worker. The worker agent sends the result to the delegate agent. Delegate agent sends the result to the manager agent, then the manager agent reports results to the client delegate which has submitted the task. The client delegate sends the result to the client which is owner of the task.

3.2.2. Task Migration Protocol

The task migration protocol works as follows. The migration manager agent analyses the communication between the tasks. The migration manager agent reports to the grid manager agent that which task should move to which container to reduce the network traffic. Manager agent locates an available worker agent in the desired container which can accept the task. The migratory agent, which is the agent that runs the task will migrate after migration analysis, is informed with the identifier of the migration acceptor agent. The migratory agent pauses the task and sets the class member *isMigrated* of the task, and sends the migration task to the available worker. The worker which gets the migrated task becomes ready to resume the task.

3.2.3. Task Resuming Protocol

The task resuming protocol is described as follows. Worker agent creates a task runner object which creates the thread that will resume the migrated task. The task runner runs the migrated task. The task checks its *isMigrated* class member. If it has been set, the task resumes its job, else the task starts from beginning.

3.2.4. Task Messaging Protocol

Tasks can interact with each other by sending or receiving massages. We defined the task messaging protocol as follows. Task constructs a serializable task.TaskMessage message with required parameters while task is running. The sender task sends the message to the task runner object. The task runner sends the message to the worker agent. Worker agent embeds the serializable task message as content object parameter of a standard ACL message of Jade environment. Worker agent also sets the receiver agents' identifiers of the message to the ACL message. Then sender worker sends the ACL message to the specified agents which run the receiver tasks of the task message. At the receiver side, the message receiving action is implemented using a message handler which is based on Jade behaviours. The message handler is implemented as a Jade behaviour which handles the messages that an agent receive. We add that behaviour to the worker agents. The message which is sent from a sender task/agent is received from the message handlers of the receiver agents. The message handler passes the received ACL message to the receiver agent. The receiver agent sends the ACL message to the task runner. The task runner gets task message from content object of the ACL message and submits the message to the receiver task.

3.2.5. Messaging Reporting Protocol

The task messaging information should be reported to the migration manager agent to maintain messaging based migration. The protocol of this reporting action is described at the following. Each worker agent has hash lists about the messaging which it realized. The hash lists have the messaging task identifiers as the key values. The values of hash list the counts of the sending or receiving messages for each messaging task. Each worker updates their messaging list during task execution. Workers send the task messaging information to the migration manager at specified arrival of time. The migration manager collects and records the messaging information which are sent from workers.

3.2.6. Migration Analysis Protocol

The migration manager agent calculates the migrations to be done from the messaging information which is gathered from the workers that executes messaging tasks. After migration analysis the migration manager agent reports the migration decisions to the grid manager. The grid manager reports the migration decisions to the workers to realize the decided migrations. The migrations is performed by the informed workers with (providing) strong mobility. Strong mobility is achieved using Brakes library. The migration analysis algorithm which the migration manager uses is implemented using the Jacop constraint solver.

3.3. Migration Analysis Algorithm

We assume that each container works on physically different machine in the grid system. The migration analysis algorithm aims to reduce the network traffic load via migrating the frequently messaging tasks to the same container/machine. We used the Jacop constraint solver to implement the migration algorithm. We formulated the loads of the grid as weighted CSP constraints and imposed each load of the system as constraints to the solver.

The migration analysis algorithm is generated as follows. Firstly we create a store object to impose all the loads/constraints onto using Jacop. Then we create our CSP variables which represent the tasks of our grid system. We assign the values to the variables which can take in the CSP search operation. The variables are the containers (machines). The true value-variable (task-machine) matching operation which will reduce the inter-machine messaging is our expectation from the CSP solving calculation. So we impose the inter-machine messaging count of each task, which the information is gathered from workers to the migration manager during task executions, as a cost for calculation using weighted constraints. So increasing of inter-machine messaging count of tasks increases the total cost at the same rate. Then we add the possible migrations of tasks to another machine to the calculation as another weighted constraint. Namely if an agent moves on another machine the system takes a task moving cost and adds it to the total cost function. Then we impose the strict relations of tasks to machines as constraints. For example if a task needs to a specific resource, the algorithm constraints that task to the machines which supplies that resource.

Thus the load balancing decision problem of grid becomes a weighted CSP solving problem. We used a generic CSP solver library called Jacop which is implemented in Java to solve our migration analysis problem as WCSP. After execution of CSP problem Jacop generates value-variable assignments. These assignments are task-container (machine) matching which will reduce inter-container messaging count for our grid system. Then the grid applies the decisions of Jacop calculation on grid. If a task has to move another location on grid, the migration manager of grid arranges the migration of the task to the determined location.

3.4. Applying Strong Mobility

We mentioned that we applied the strong migration of the tasks that are running on the grid. Strong mobility can be provided by various mechanisms. These mechanisms can be generally implemented in four levels of programming concept; these are operating system level, interpreter level, source code level, compiled code level. Each level has trade-offs between efficacy and portability. The approach we used is compile code level which let us to work on standard JVMs. To be able to migrate a thread, its execution must be suspended and its stack and program counter must be captured in a serializable format which is then going to be sent to the target location [11].

We used the Brakes library to implement the strong mobility. Brakes has a bytecode transformer which instruments Java classfiles so they can capture their internal states at any time. Basically Brakes can rewrite a Java classfile with including computation state information. That allows us

to pause and resume Java threads whenever desirable. In this paper we used Brakes thread serialization technique to apply compiled code level strong mobility approach in our grid. All the tasks which is run on grid is rewrote at compiled code level to achieve strong mobility.

The grid environment which we used is based on JADE[12]. The standard messaging infrastructure of JADE allows sending context objects which has implemented serializable interface of Java. We designed our tasks as implementation of the serializable interface. The tasks also have the computation object which is provided by Brakes. Computation object keeps the execution state information. The overall mechanism which we told about allows sending a task with the captured internal state from one location to another on the grid. The sent task is resumed at the target location by the assigned worker.
The mechanism works as follows:

- The migration manager report the migratory worker agent of task to the migration acceptor worker agent which run the task that will be migrated,
- Migratory worker captures the task execution with Brakes,
- Migratory worker serializes the task and execution state with standard Java serialization,
- Migratory worker sends the task to the migration acceptor worker which is assigned by the migration manager with a JADE message,
- Migration acceptor worker deserializes the task and resumes it with Brakes.

The migration manager informs the worker agents after migration processes about changes of the locations of the migrated tasks. So the task which is migrated to another location and its messaging task can move on messaging each other.

4. EXPERIMENTS AND RESULTS

We have implemented virtual environments which have tasks that are messaging each other on AGRID. Also we can test the migration algorithm separately from grid environment with a test tool. The test tool can emulate data which our migration algorithm needs from grid. The migration algorithm can solve the real world grid data and the artificial grid data which is generated from test tool with the same way. So we applied two types of tests, one of them is creating a real grid which has real tasks that have resource needing, realizes messaging of tasks with each other, contains multiple real machines and achieves real task migrations from a machine to another. The other test type is emulating a big sized grid environment data, which has virtual machines, messaging of tasks, resource needing of tasks, count of workers and clients. Both test types the algorithm makes decisions the same way.

In the first experiment we set a real grid environment which had two machines and four tasks and five workers. “Figure 2.” shows the first environment. The tasks are initially assigned to workers by the manager agent considering their resource needing. But at the first case the manager agent does not have information about messaging between tasks because the task messaging is happening dynamically and unpredictably during the tasks’ life time. So we can say that the initial assignments of tasks to workers are random except for resource needing. After a certain time the migration manager collects the messaging information, runs the migration algorithm and decides the migrations. The migration manager decided to migration by performing the following calculations. The total cost of the environment was 32 (Task1-Task2 inter-messaging) + 49 (Task3-Task4 inter-messaging) = 81 before the migration. Then algorithm analyzed the total cost of the all possible migrations. The only migration which the first environment can perform was moving T2 to the Machine2 because of resource needing. The total cost of this migration was 49 (Task3-Task4 inter-messaging) + 9(Task2-Task4 inter-messaging) + 20(an empiric task moving cost for the environment) = 78. So the algorithm selected the least total cost then decided to move Task2 to Machine2.

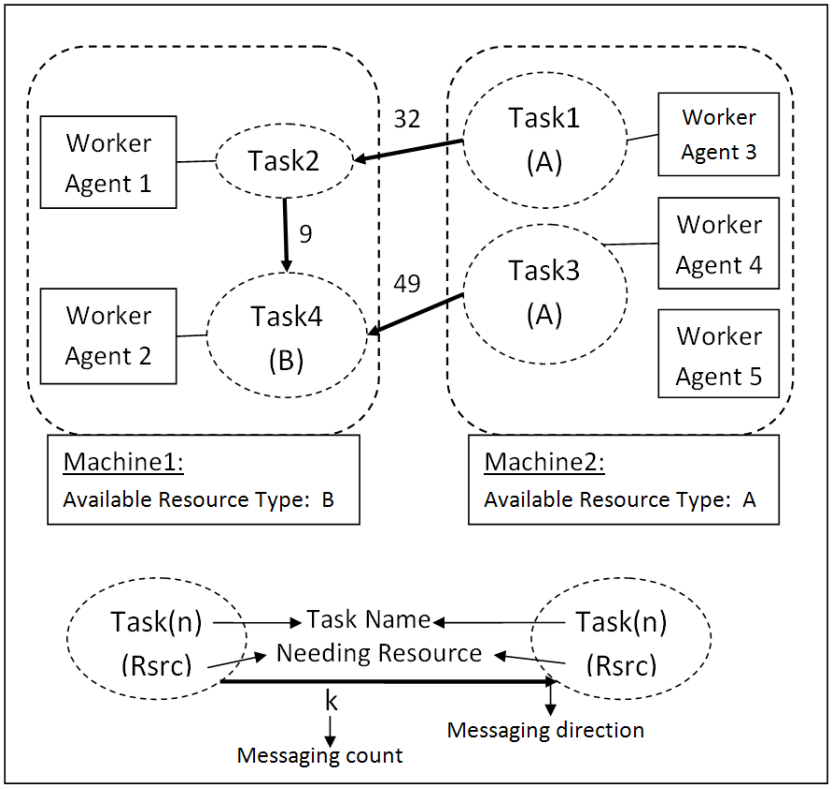

Figure 3. The first environment

The other experiments are performed with our test tool. We can generate different environments using our test tool. Each environment we can set messaging factor, resource needing factor and moving cost value as parameters to generate different environments. High messaging factors leads to generate highly messaging tasks, high resource needing leads to generate highly resource needing tasks. The moving cost can be set to intended values for each environment. We can see the effects of the changes of those factors over results.

We generated second environment using our test tool. This environment had 40 machines, 8 workers for each machine, 4 tasks per worker and 19 as move cost. We observed the effects of changing of messaging factor and resource needing factors in various experiments. All experiment results are average of ten experimental results. We used RND as Resource Needing Factor, MF as Messaging Factor, MGC as Migration Count, T as calculation Time, MCBM as Messaging Count Before Migration, MCAM as Messaging Count After Migration, MVC as Move Cost and MRR as Messaging Reduce Rate at the graphics of experimental results.

Table 1. Results of MF=0 at the second environment

RND	MGC	T	MCBM	MCAM	MRR(%)
0	26	0.11	1771	494	72.11
3	22	0.06	1771	589	66.74
5	18	0.05	1771	878	50.42
8	17	0.03	1771	951	46.30
10	9	0.03	1771	1401	20.89
13	6	0.02	1771	1514	14.51
15	4	0.02	1771	1619	8.58

Table 2. Results of MF=5 at the second environment

RND	MGC	T	MCBM	MCAM	MRR(%)
0	36	0.11	3229	1441	55.37
3	41	0.24	3229	1264	60.85
5	33	0.29	3229	1628	49.58
8	31	0.13	3229	1592	50.69
10	27	0.09	3229	2030	37.13
13	9	0.05	3229	2705	16.23
15	3	0.01	3229	3073	4.83

Table 3. Results of MF=10 at the second environment

RND	MGC	T	MCBM	MCAM	MRR(%)
0	62	5.54	5834	2537	56.51
3	59	6.33	5834	2751	52.84
5	48	5.86	5834	3112	46.66
8	38	5.56	5834	3773	35.32
10	38	6.72	5834	3446	40.93
13	24	5.94	5834	4394	24.68
15	12	5.34	5834	5227	10.40

At the third experiment we tested the effects of changes in move cost over results. We had 40 machines, 10 workers per machine, 6 tasks per worker at third environment.

Table 4. Results of MF=0, RND=0 at the third environment

MVC	MGC	T	MCBM	MCAM	MRR(%)
5	39	0,25	2258	652	71,12
10	37	0,20	2258	672	70,23
19	34	0,25	2258	616	72,71
35	23	0,21	2258	914	59,52
40	21	0,21	2258	987	56,28
50	15	0,23	2258	1250	44,64
75	6	0,21	2258	1786	20,90

Table 5. Results of MF=5, RND=5 at the third environment

MVC	MGC	T	MCBM	MCAM	MRR(%)
5	68	1,18	4909	2315	52,84
10	64	3,73	4909	2315	52,84
19	57	18,01	4909	2232	54,53
35	44	98,54	4909	2463	49,82
40	39	119,07	4909	2584	47,36
50	30	59,59	4909	2977	39,35
75	8	59,09	4909	4263	13,15

The second environment we can see the increasing of the resource needing factor reduces the migration count and messaging reduce rate, the increasing of messaging factor increases the messaging count and migration count but there is no significant effect of messaging reduce rate. The third experimental environment tells us increasing of move cost reduces the migration count and messaging reduce rate.

5. CONCLUSION

In this paper we used weighted CSP approach for migration analysis on a grid environment. We presented a dynamic solution to implement different migration decisions. We can change easily the migration analysis algorithm by adding new loads for the grid system as constraints to the CSP problem. Adding new constraints changes the migration analysis algorithm and the migration decisions totally. For example if we add the processing power of each machine as a load for the grid, the algorithm will work as processing load balancer.

All grid components, algorithms and the applied migration approach are fully implemented in Java and the grid system can work over standard JVMs. This makes us our approach platform independent.

The future work will be improving our approach dealing with the flexibility of the algorithm changes. Implementing an interface that will be plugged into the algorithm externally allows us changing the algorithm without modifying the core implementation of migration algorithm. This also makes our approach re-usable.

REFERENCES

[1] U. Gumuş, N. Erdoğan, "AGRID- Agent Based Grid System," 1st International Workshop on Infrastructures and Tools for Multiagent Systems (ITMAS Workshop at AAMAS 2010), Toronto 2010.

[2] E. Truyen, B. Robben, B. Vanhaute, T. Coninx, W. Joosen, and P. Verbaeten, "Portable support for transparent thread migration in Java," In Proceedings of the Second International Symposium on Agent Systems and Applications and Fourth International Symposium on Mobile Agents (ASA/MA2000), volume 1882 of Lecture Notes in Computer Science, pages 29–43, Zurich, Switzerland, September 2000. Springer-Verlag.

[3] F. R. L. Cicerre, E. R. M. Madeira, and L. E. Buzato, "Structured process execution middleware for grid computing: Research articles. Concurr. Comput. : Pract. Exper., 18(6):581–594, 2006.

[4] B. Yagoubi, Y. Slimani, "Dynamic Load Balancing Strategy for Grid Computing", Transactions on Engineering, Computing and Technology, volume 13, May 2006, pp. 260-265.

[5] Aversa R, Martino BD, Mazzocca N, Venticinque S. MAGDA: A mobile agent based grid architecture. Journal of Grid Computing 2006; 4(4):395–415

[6] C. Ansótegui, M.L. Bonet, J. Levy, F. Manyà, "The logic behind weighted CSP," in: Proc. of the 20th Int. Joint Conference on Artificial Intelligence, IJCAI'07, AAAI Press, 2007, pp. 32–37.

[7] R. U. Payli, E. Yilmaz, A. Ecer, H.U. Akay, and S. Chien, "DLB A dynamic load balancing tool for grid computing," in Procs. Parallel CFD04, G. Winter, A. Ecer, F. N. Satofuka, P. Fox (eds.), Elsevier (2005), pp. 391–399
[8] M. Li, M. Baker, "The Grid Core Technologies", John Wiley & Sons Ltd., 2005
[9] C.-Z. Xu and F. Lau, "Load Balancing in Parallel Computers: Theory and Practice," Dordrecht, Germany: Kluwer, 1997.
[10] K. Kuchcinski and R. Szymanek. "JaCoP Library. User's Guide," http://www.jacop.eu, 2009.
[11] T. Coninx, E. Truyen, B. Vanhaute, Y. Berbers, W. Joosen, and P. Verbaeten. "On the use of threads in mobile object systems," In 6th ECOOP Workshop on Mobile Object Systems, Sophia Antipolis, France, June 2000.
[12] G. Caire, "Jade Tutorial-Jade programming for beginners," TILab, 2003, http://jade.tilab.com/doc/tutorials/JADEProgramming-Tutorial-for-beginners.pdf
[13] N. Suri, J. M. Bradshaw, M. R. Breedy, P. T. Groth, G. A. Hill, R. Jeffers, T. S. Mitrovich, B. R. Pouliot, and D. S. Smith. "NOMADS: toward a strong and safe mobile agent system," In C. Sierra, G. Maria, and J. S. Rosenschein, editors, Proceedings of the 4th International Conference on Autonomous Agents (AGENTS-00), pages 163{164, NY, June 3{7 2000. ACM Press.
[14] R. S. Gray, G. Cybenko, D. Kotz, R. A. Peterson, and D. Rus, ''D'Agents: Applications and performance of a mobile-agent system,'' Software – Prac. Exp., vol. 32, no. 6, pp. 543 – 573, 2002.

AUTHORS

Zafer Altuğ Sayar received his B.S. in Computer Engineering at Engineering Faculty of Istanbul University in Istanbul, Turkey. He obtained his M.Sc. in Computer Engineering at Computer and Informatics Faculty of Istanbul Techical University in Istanbul, Turkey. His current research areas are grid systems, multiagent systems and mobile agent systems. He has been employed as a researcher in The Scientific And Technological Research Council Of Turkey (Tubitak) since 2008.

Nadia Erdoğan received her B.S. in Electrical Engineering and M.Sc. in Computer Science Department of Bosphorus University in Istanbul, Turkey in 1978 and 1980 respectively. She received her Ph.D. in Computer Engineering Department of Istanbul Technical University in Istanbul, Turkey in 1987. Dr. Erdogan is a professor in the Computer Engineering Department of Istanbul Technical University, Istanbul, Turkey. Her current research areas include distributed computing and execution environments, mobile agent systems, multiagent systems and parallel programming.