Journal of Universal Computer Science, vol. 15, no. 4 (2009), 941-966
submitted: 30/6/09, accepted: 27/2/09, appeared: 28/2/09 © J.UCS

Agent Migration: Framework for Analysis

Dariusz Król
(Institute of Informatics, Wrocław University of Technology, Poland
Dariusz.Krol@pwr.wroc.pl)

Aleksander Lupa
(Institute of Informatics, Wrocław University of Technology, Poland
133689@student.pwr.wroc.pl)

Abstract: A lot of work is devoted to analysing architectures for coordinating the behaviour of individual agents. However, providing agents with abilities to migrate continues to be a highly challenging problem. We propose a novel multi-agent framework, called *Agent-based Migration* (AM). We begin by defining the principal objective, which is the migration phenomenon applied, in our case, to distributed calculation of prime numbers. We present the AM architecture in detail. Then, we introduce different types of migration and the communication scheme. We also conduct a set of experiments in two environments: 4 heterogeneous computers and 45 (almost) homogeneous computers. Specifically, we are looking for a way to find optimal configurations for migration in both environments. The conclusion from this work is that introducing propagation to a system in a form of agent migration in both networks could considerably decrease the execution time according to used algorithms and established assumptions.

Keywords: Propagation, Mobile Agent, Multi-Agent System, Performance Evaluation
Categories: C 2.4, C.4, H.1.0

1 Introduction

The efficient utilization of network resources is an important issue. The problem is hard due to the distributed nature of computer networks, high communication demands and the desire for limited communication overheads. One solution to such challenges is to design efficient, decentralized and fault-tolerant data propagation model which accomplishes tasks with no access to global network information. Mechanisms of agent propagation are useful because agents can be organised into efficient configurations without imposing external centralised controls. Operation without a central coordinator eliminates possible bottlenecks in terms of scalability and reliability. Other most interesting properties of modern networks are mechanisms of biologically inspired self-organisation [Chakravarti, 05].

The concept of information propagation is common to distributed environments in the following forms: data propagation for collaboration [Kazienko, 07] [Mitschang, 03] [Repantis, 04], adaptive task allocation and resource negotiation [Fatima, 01] [Jiang, 05], mobile object and code update [Levis, 04] [Navvaro, 05], message dispatching [Korzeniowski, 08], routing algorithm [Tan, 03] [Wedde, 06], self-organising computation [Bernon, 06], recommendation [DZhang, 08] and many more.

In spite of the rapid growth of complex networks [Centola, 07] and multi-agent systems [Ni, 08], there is no universal measurement method that allows evaluating the

performance of data propagation processes. The existing works on systems performance evaluation deal principally with the structural aspect concerning the network communication topology [Jurasovic, 06] [Krol, 08] [Zhang, 08], with the syntactical aspect concerning the message complexity and critical thresholds [Huang, 08], and eventually the statistical aspect concerning the quantitative methods to analyse robustness (the ability of a system to perform without failure under a wide range of conditions) [Krol, 09]. Like [Hmida, 08] our study covers all three aspects, but in contrast to his model, we place particular emphasis on the flexibility, the ability of a system to readily adapt to changing requirements via agent propagation. The analyses generally focus on three parameters: the network communication complexity, e.g. performance metrics, the time needed for operation completion, e.g. round trip times, and the traffic generated during execution, e.g. data transferred and message overhead [Chmiel, 05].

The contribution of this work lies in the fact that by data propagation in a network (provided by agent migration) we reduce execution time. Agents with their ability to react on changes in the environment, i.e. changes in the number of agents on a node, are able to adapt to current conditions using migration. Available resources can be adjusted to needs of computation by migrating agents using a load balancing algorithm [Cao, 05]. There are also a few additional research objectives. The first one is the investigation of different migration types, if and how they affect the execution time. The second is finding the optimal configuration of parameters for the computed task in two different networks and finally, discovering the dependencies between key features of the experiments like migration type and execution time.

The remainder of this paper is structured as follows. We begin by explaining the concept of migration found in the distributed computing. In Section 3, we present the AM architecture in detail. We then introduce different types of migration and the communication scheme. In Section 5 the experiments with a discussion of the results are presented. Finally, the last section summarizes the work.

2 Migration Phenomenon

2.1 The Problem

With the growing number of computers connected in a network there are more possibilities to distribute information in a flexible way. When the number of nodes in the network becomes too high, the centralized control is becoming difficult, inefficient and the system does not scale. The communication costs within big centralized network are becoming high and time-consuming. The time to broadcast a message from one computer to the set of others and getting the acknowledgement in such environment is unacceptable. The conclusion here is that the coordination of objects involved in distributed problem solving has to be done generally in a decentralized way. Objects should be able to migrate in order to increase the efficiency of the system. An important fact here is that the coordination of work in a decentralized network demands high level of autonomy from those objects [Krivokapic, 00].

Objects are composed of a state, data and a set of operations that can be invoked from outside in order to manipulate that data. This is the idea of a passive object, which does what it is asked without any question. Active objects are equipped with

autonomous behaviour – so they are not controlled from outside, e.g. by method invocations, but they have the ability to perceive external environment and react to changes occurring in it. In many cases such objects would be large and they would have a set of methods, which allow operating on their data. When many objects are involved in a common task, they need to cooperate. Different tasks require different kind of cooperation (also in intensity). In order to do it they have to communicate by means of messaging and decide all over what action, if any, should be taken.

Object migration has to meet certain requirements:

- *Transparency and availability*. The migration process should not be visible to the user. Also when an object is moving, it should be available. For example, messages sent to the object during its movement should be received.
- *Fault tolerance*. Migration always results in a correct state. The input of the process is an object at location 1 and the output is the same object in the location 2. There are no additional copies of the object. Even if there are failures in the network, the object should not be lost.
- *Flexible migration policy*. Migration algorithm should be exchangeable over time. Migration policy should be sometimes upgraded to adjust to the changing environment. There has to be a possibility to tell a certain object to move from one location to another because of planned actions, security reasons or errors in the network.
- *Efficiency*. The migration, including the decision and the process, has to be efficient. An object should not migrate too often, because of the costly decision making. After migrating to another location, it should stay there long enough for a new evaluation.

The implementation of migration is based on the premise that the object is not executing operations during the movement. The migration process has to be executed after agreement of two locations, before the action is made. If the object is going to migrate first it will ask if the migration is possible. There should be a local manager on a location where the object currently is deployed and where it is going to be moved. Both managers have to agree to the migration process. Moreover, there should be another object, which would be responsible for instantiating the migrating object at a current location. The details of proposed migration process are described in the next section.

If an object has been moved incorrectly, the communication costs and transaction time conducted using this object may increase. When objects are moving in the network, the environment becomes dynamic and sometimes it is very hard to predict what will happen after changes. So it could be expected that one object will be moved many times. The decision about migration is made by the object and it is based on information about the network.

Beside migration there is a possibility of cloning an object. Cloning means that the original object stays in the origin environment, but the copy could be created in another environment (or in the same environment). If there are global object identifiers, the new born object has to have other identifier than the original one.

The description of an object in this section is very close to the agent definition. Here the object is autonomous; it has to cooperate with other objects. It is also reactive and in some way proactive, because it decides when to migrate.

2.2 Prime Number Generator

The prime number generation problem was selected for this research. We assume that we generate a given number of primes above a given start number and write the results into the file. All computers search prime numbers and as soon as possible they send the solution to the one, which is responsible for saving the results in a file. Prime number generator was selected to be solved in a distributed way because of its computation-focused nature. A range of numbers to examine is given and from the result point of view, it does not matter, on which machine the computation is running. The most important aspect is that there is a possibility to move agents from the slow computers to the faster ones.

The algorithm for prime number is simple. For each x from given range $[a,b]$ calculate square root $c = sqrt(x)$ and check if any number from range $[2,c]$ divides x without reminder; if no then add x to the list of prime numbers. Of course, there are more efficient ways to calculate prime numbers. The goal is not to calculate them as fast as possible, but to show how distribution could be managed using agent migration.

This approach has several advantages. One of them is that the range to search the primes can be divided, and the results can be merged at the end giving the complete solution to the problem. In order not to create a bottle-neck, partial results are being sent continuously to be saved into the file.

3 Agent-based Migration Framework Architecture

In this architecture, in general the AM framework consists of two kinds of units: nodes and a broker. A node is a component, on which agents reside, which is supposed to search prime numbers. A broker is a component, which distributes the task into the framework and saves the results into the file. It is not destined to calculate primes.

We assume the following algorithm for distributing the task over the nodes. When the broker gets the task order it knows how many nodes are available, because when a node enters the network, it sends a ready message to the broker. We do not consider any changes after the broker has received the task order. At the beginning there is no information about the available resources, so primes search range is divided equally by the number of nodes and then sent as a task request. While processing, nodes also send found primes to the broker by portions.

This section describes how tasks in the framework are accomplished, which agents are responsible for carrying them out and what the structure of messages connected with these processes is.

3.1 Main Process

The main process is the core of the framework. Its main goal is to distribute task and afterwards collect all messages concerning reporting in order to display the final results. All activities used in the diagram are agent services.

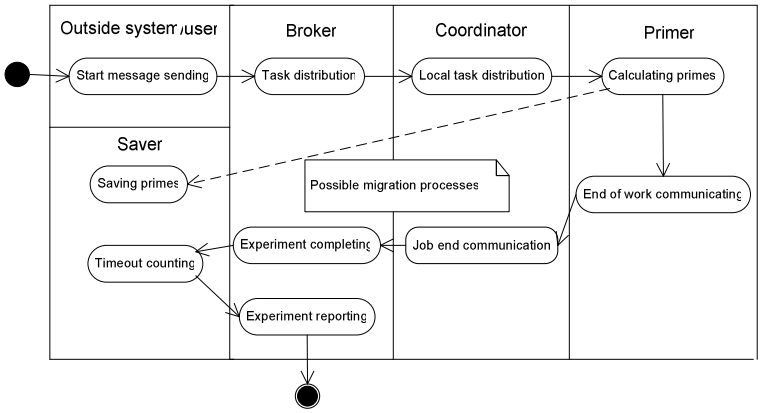

Figure 1: Main process diagram

There are 4 types of agents participating in the main process: Broker, Saver, Coordinator and Primer. This process, depicted on Figure 1, starts outside of the system, when a user or agent from another system sends a message to the Broker that includes the range of numbers to search primes from. The next step is done by the Broker, which distributes task to Coordinators. The diagram shows the control flow only for one Coordinator and one Primer, with actually many Coordinators and many Primers.

Task distribution from the Broker goes to the local level and finally each Primer gets its task. Searching primes consists of many single search processes after which results are sent to the Saver agent. This is shown in figure by a dashed arrow. The box "Possible migration processes" denotes the place in the main process flow when migration process is possible. They take place after the initial task distribution and before end of work.

The last phase of the main process starts when Primers finish searching the whole range of numbers they got. Each of them sends a message to the Coordinator (a Coordinator "commanding" the location where Primer resides) and when all Primers report back to Coordinators, agents send job end message to the Broker. When Broker gets all job end messages, it expects the main process to be nearing completion. Because there are possible message asynchronisms, Saver agent starts a timeout counting in order to receive all messages from Primers, whose results have not been written into the file so far. This possibility exists mainly because each result message includes large data to write. When timeout is over, Saver agent confirms that there are most probably no other result messages (also called save messages). When Broker gets this message, it displays information about the experiment. This ends the main process.

3.2 Migration Sequence Diagram

The sequence diagram for the main process shows all messages that are sent in the system. As depicted in Figure 2 we defined 6 entities. There are only one Saver agent and Broker agent. However, in the whole AM framework there are usually many

Coordinators and Primers. In the diagram, it is denoted in a symbolic way: There is Primer 1 and Primer M, and also Coordinator 1 and Coordinator N. They reflect the synchronised and asynchronised communication patterns between agents.

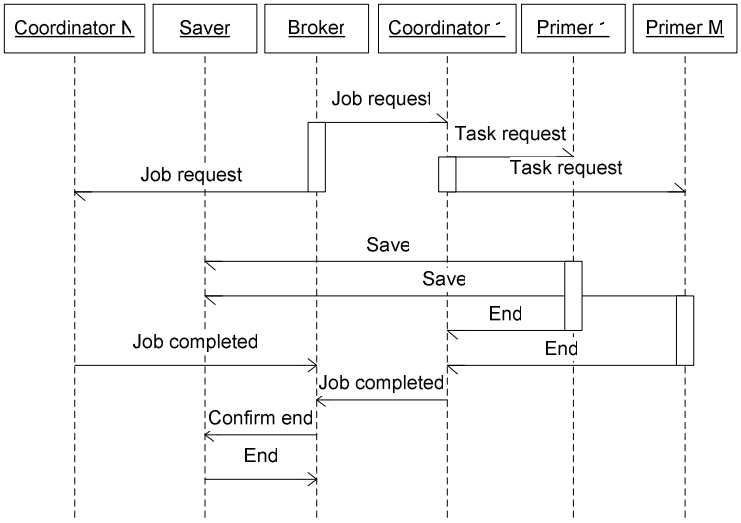

Figure 2: Migration sequence diagram of main process

The main process starts with *job request* message from Broker. It is a message with a range of numbers to search primes for each node. These job requests are sent by Broker agent asynchronously. After a Coordinator gets a *job request* message it distributes tasks locally by sending request to Primers also asynchronously. In the meantime a message from Broker to another Coordinator (e.g. Coordinator N) might have just arrived. Hence when some Primers might have already started searching, on another node the job message might have just arrived.

Primers after checking some numbers send them to the Saver agent and then there is a possibility of a problem. When a Primer is ending its work it sends asynchronously a *save message* to Saver and *end message* to a local Coordinator. It is important to mention that agents on each node are working asynchronously. After a Coordinator gets *end message* from all Primers, it sends *job completed* to the Broker. When a Broker gets *job completed* message from all Coordinators it sends a *confirmation message* to the Saver agent. And as we can see in Figure 2, there is one problem. It is shown for Primer 1 and Coordinator 1. Sometimes it happens that a *save message* sent by Primer 1 arrives later than three messages sent in a sequence: *end message* from Primer 1 to Coordinator 1, *job completed* message sent by Coordinator 1 to Broker and *confirm end* message sent from Broker to Saver. That is the reason of timeout process done by Saver. When timeout elapses, it sends *End* message to Broker, which also ends the entire processing.

3.3 Migration Process

The migration process diagram presented in Figure 3 is an activity diagram with three classes of agents: Migration Coordinator, Coordinator and Primer. They represent what happens when an object flow (a message) arrives – how the states are changing.

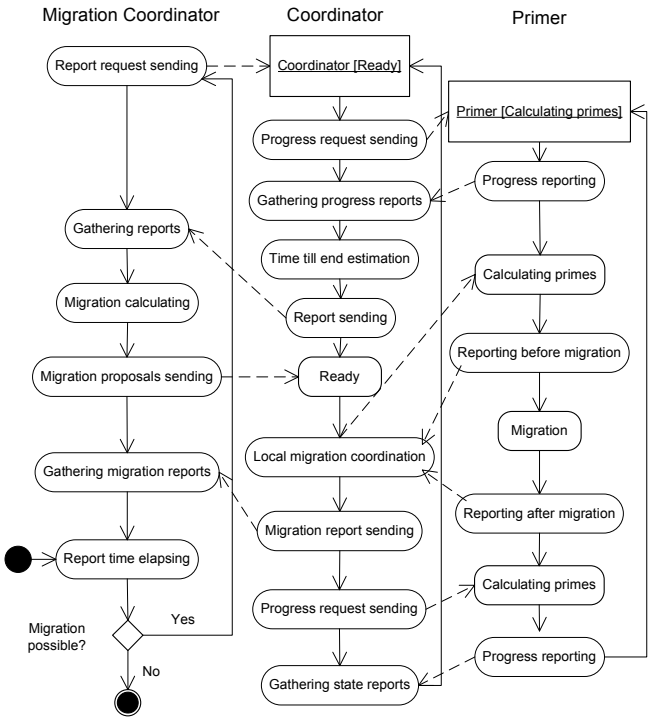

Figure 3: Migration process diagram

Migration process starts from Migration Coordinator. When all agents are ready just after the start of an experiment Migration Coordinator also gets a message from Broker that experiment has started. After this event it starts counting the report time. After this time is over, it has to check if a migration is possible. There are few migration conditions. One of course is that at least two agents are still active and they have estimated their time for more seconds than the node threshold (time till end on a node after which migration is not applicable). When this is true, the whole process starts. If not, the task is already finished or will be finishing soon.

After report time is over, Migration Coordinator goes to state when it sends report request messages to all Coordinators participating in the experiment. Afterwards it changes state for gathering those reports. In other words, it waits for all reports to arrive. When a Coordinator gets a message to report, it sends progress report request to all Primer agents in the current location. Afterwards, like the Migration Coordinator, it changes state for gathering those reports. When a Primer agent gets this message, it is calculating prime numbers, so it checks how much it has done and reports back to Coordinator. Afterwards it returns to accomplishing its task.

When Migration Coordinator sends all migration proposals, it goes to the state of gathering migration reports. In other words, it waits for Coordinators to report to it that all migrations are completed. A Coordinator agent always knows how many agents should depart or arrive. After getting a migration proposal message from the Migration Coordinator with the destination where to migrate, agent sends a migration proposal message to the first available active agent (Primer that has not reported end of work). When a Primer gets this message, it stops calculating and prepares itself for a migration. Just before leaving and just after arriving in another environment it sends a message to the local Coordinator with the information about the range that it was checking and the range of numbers that it has already checked. In this way Coordinator always knows how many numbers are checked for being primes and how many are to be checked. After a Primer settles down in the new environment it goes back to its main task – searching primes.

There is a possibility that a migration proposal will be rejected. This might happen when some changes will take place in time between report sent to Migration Coordinator and migration proposal message delivery. This situation happens when some Primers are finishing their job and reporting it. Then a Coordinator will move them to the non-active group – so they cannot migrate. When in the migration proposal it is suggested that the last active agent has to leave the node, it will be rejected, because node with no active agent when the experiment is not finished has no value of existence.

When a Coordinator notifies that there are no incoming or departing agents left, it sends a message to the Migration Coordinator that local migration is finished. When the Migration Coordinator gets this kind of report from all Coordinators that were supposed to report, it starts counting the report time and the algorithm starts again.

3.4 Migration Calculation

This process assesses the ability of a node to perform a task. The key point here is that the estimation is based on the previous efficiency of a node when performing a task. It is compliant to the assumption that at the beginning of an experiment the ability of nodes to perform the task is unknown.

When a Coordinator gets all progress reports it estimates remaining time till the end of experiment based on the data it has. This is an important moment in the whole algorithm. Time till the end is estimated based on a sample from the previous change in the environment for the node – from the last migration or the beginning of the experiment. Basically, when Primers are reporting, they deliver two parameters - what is the range of numbers to examine and how many numbers have already been checked. All Primers report that there are two sums being calculated: a sum of numbers to check and a sum of numbers that already have been examined. Based on the new and old report there is a quantity of numbers calculated that have been examined. Then, the time from the last change is calculated. Based on these two values the *speed* for node i (1) is calculated.

After the Migration Coordinator gets all reports form Coordinators it starts the calculation. During that time all Coordinators wait for messages. When the agent network is being created at the beginning of an experiment, Migration Coordinator creates a list of objects describing nodes. This list is used for migration calculation but also for remembering names, locations in the network and for information if a node

has to report back after migration finishing. As the reports arrive the information in the list is being refreshed.

$$speed_i = \frac{currently_checked_numbers_i - last_checked_numbers_i}{current_time_i - last_change_time_i} \quad (1)$$

$$estimated_time_i = \frac{number_of_numbers_to_check_i}{speed_i} \quad (2)$$

$$agent_value_i = \frac{estimated_time_i}{number_of_agents_i} \quad (3)$$

$$agent_change_i = \frac{average_active_time - estimated_time_i}{agent_value_i} \quad (4)$$

$$proposed_agents_i = number_of_agents_i + agent_change_i \quad (5)$$

$$norm_proposed_agents_i = proposed_agents_i * \frac{current_agents_sum}{sum_of_proposed_agents} \quad (6)$$

For each node we have data on the *estimated time* (2) and the list is sorted beginning with the shortest time till the end of experiment. Then for each node three values are calculated. The *agent value* for node *i* (3) is a measure of how much time from the *estimated time* till the end falls to one agent. The next value is a bit more complicated (4). In this algorithm there is such a value as an average active time. The term active time applies to those nodes only, where migration can take place or in other words, which have the estimated time higher than the node threshold parameter (see Section 5.1). So the goal is to calculate how many agents on node *i* should have the time as close to average as possible. The assumption is that an agent (Primer) represents some work to do and if there was a certain number of agents, then the time would be close to average. Having such simple assumption, the number of proposed agents for each node is calculated (5). If for example a node has a time lower than average – then there should be more agents and the agent value change is greater than zero. If not then some agents should migrate from this node. But after calculating the proposed agent number there is a possibility that there should be more or less agents than currently is, so it is necessary to correct this number on each node by sum of agents divided by sum of proposed agents (6).

After this process agents are distributed according the resources (agents) available in the system. But there is a possibility that still the sums of agents and proposed agents are not equal, so there is correction algorithm, that makes these sums equal by adding or subtracting proposed agents for each nodes starting from those that have the biggest number of proposed agents.

After executing this algorithm a list of proposed migrations is created. Building this list is based on making equal the number of agents and proposed agents on a node (in the node information list) possessed by Migration Coordinator. Agents always migrate from the node that has the biggest estimated time to the node that has the lowest estimated time.

4 Migration Types and Communication Scheme

JADE [Bellifemine, 08] [JADE, 08] a mature and the most popular multi-agent platform was selected to be the environment for our framework implementation. One of the main reasons in its favour is that it is used in many commercial and industrial applications and it is considered as an efficient one. It is also widely accepted by scientists who test their project implementations on JADE. As many other muli-agent platforms, JADE is compliant with FIPA specifications and it is released under an open source licence.

4.1 Migration Types

There are three migration types: move, copy and birth. Their differences in dispatching messages were depicted in the migration sequence diagram. These three ways are different, because of the differences in core implementation. In the framework they are implemented using JADE methods and also by sending messages. Migration is available only for Primer agents, which actually represent the work to be done.

4.1.1 Move

As shown in Figure 4, the move migration is intended to support agent mobility and is called either by the Agent Platform or by the agent itself to start a migration process.

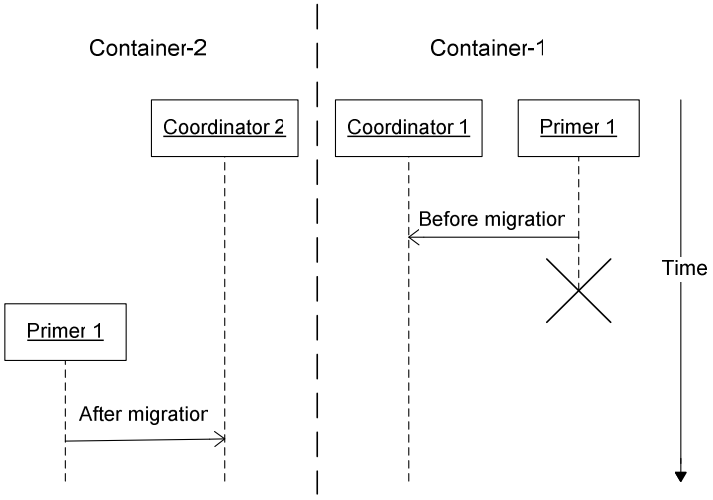

Figure 4: Move migration sequence diagram

When an agent invokes a move method itself a message is created, which contains the agent code and data. In Java programming language data is composed of asset of object. Those objects are transformed to byte array, which can be stored in a message, using the Java serialization mechanism. This message, with code and data is transferred through the network using TCP/IP protocol. Java deserialization

mechanism is used at destination to recover objects. The class loading (*ClassLoader)* mechanism allows classes incoming in the message to be dynamically loaded (transformed into executable classes). So the list of crucial processes for one-way migration process is stated below: Creation of the agent, Serialization of the agent, Message sending to the destination environment (there has to be an acting server), Deserialization of the agent at destination, Loading the code from incoming agent and resuming the agent.

The move method requires a Location Object which holds the name of the location, container identifier object and platform identifier object. These objects are necessary to start the migration process performed by *jade.core.mobility* package. What is important is mentioned in the method description – agent state is changed from Active to Transit. The method ceases all current activities and suspends agent till the platform has relocated it. The operations before and after moving can be specified using methods *beforeMove*() and *afterMove*().

In the framework these methods are used for sending messages to Coordinators on the source and destination platform in order to have the integrity of the task sustained. The message to the Coordinator when leaving includes the difference between the beginning and the end of the range of numbers, which was examined by the current agent. After a Coordinator gets this message, it subtracts this from the overall range of numbers to be searched.

When an agent is arriving at the new location it sends a message that includes the range of numbers left to be examined. Coordinator, after getting it, adds this range to the overall pool of numbers to be checked.

4.1.2 Copy

As shown in Figure 5, the copy migration is based on cloning the agent in the new location and, after the original agent is copied, destroying it. In the implemented system coping is executed by a copy method. This method is intended to support agent mobility and is called either by the Agent Platform or by the agent itself to start a cloning process. The actual cloning takes place asynchronously.

This method beside a new Location requires also a new agent name. In a JADE platform agent's name must be unique and there cannot exist two agents with the same name. The cloning process is different than moving because an agent, which was the original one, stays on the platform it was residing. Similar to the move migration there is a possibility to prepare an agent to this operation and for example save its state by overriding methods *beforeCopy*() and *afterCopy*().

Messaging in case of copy is done in the same way as in move migration. The only difference is that after the agent is copied, the original one executes self-destruction by invoking *doDelete*() method.

In the framework this migration type is technically faster and should take less time than the previous one, because an agent does not leave the environment, so there is no time wasted for this process. Creating agent on another environment involves the same operation as in the move migration. The only additional time is for destroying the original agent.

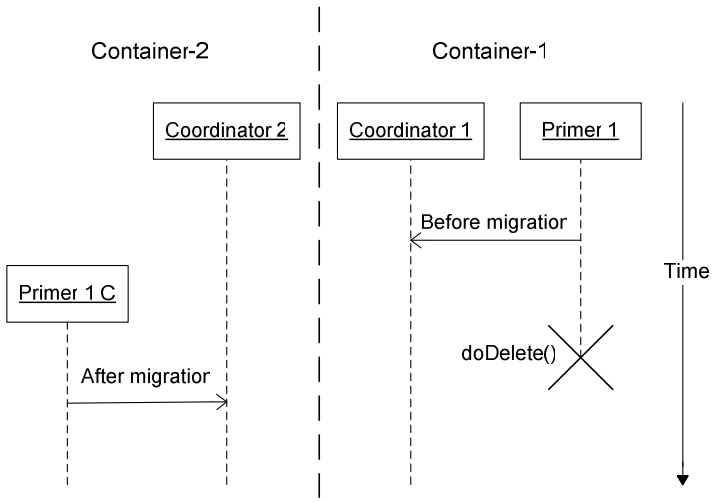

Figure 5: Copy migration sequence diagram

4.1.3 Birth

Birth migration, see Figure 6, is implemented by using messages to transfer necessary data and to activate new agent to execute its task. There is no method in JADE system that does it.

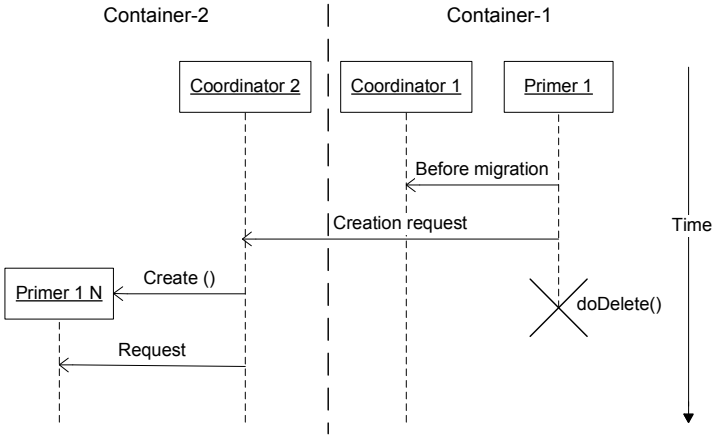

Figure 6: Birth migration sequence diagram

This process is initiated by the agent itself. The agent gets a message with the Location object, as in the previous types, but instead of putting itself to a non active state it sends a message to the Coordinator on the destined node with all necessary data to create an agent, which is the same as the original data. This data consists of the

current number that is next in the queue for checking and the end of the range of number to check. The migration type and primes range search could be added, but the assumption is that on the other node these parameters are the same.

When the Coordinator on the other node gets this message it creates a Primer agent, adds it to the list of available Primers and sends a message with the data it got before. The number of examined numbers is not necessary. After getting this request message (the same type as during the task distribution) agent starts working. The original agent executes method *doDelete*() just after sending the message.

4.2 Communication Scheme

The communication diagram presented in Figure 7 does not show all agents in the system. If there are many agents of a specific kind there are only two agents in the diagram in order to represent if there is a connection between them. The N and M suggests that usually there are a different numbers of Coordinators and Primers in the system.

When agents are building an agent network there has to be a certain sequence while doing it. This sequence was created to achieve an order also while creating agents. The second reason for creating an order lies in the set up method. While building a network, there should be a central point, which will be responsible for gathering information. Although not necessary, for this system it was easier to introduce such a central point than to add some intelligent behaviour to the agents just to recognize what other agents are in the network.

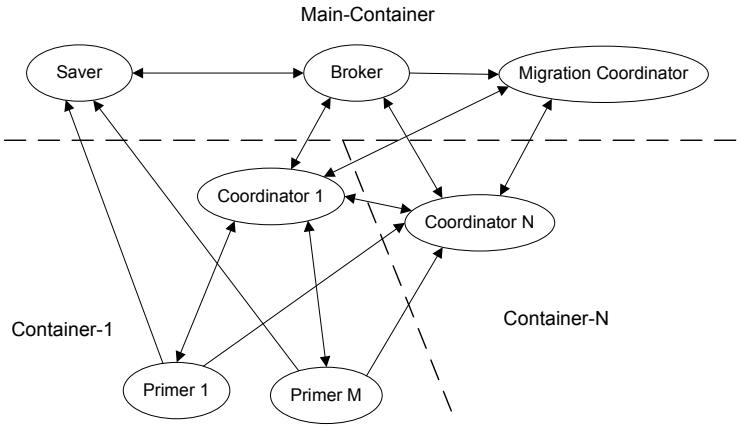

Figure 7: Communication scheme between agents

From the communication diagram it is inferred that Primers are not able to communicate directly to the Broker or Migration Coordinator agent, just to Saver, which is a simple functional agent, and to Coordinators. The arrow from the Primer to other Coordinator (not the local one) is only used when it comes to the birth migration type for the migration message. Also, Coordinators between themselves do not normally communicate – only when migration from one node to the second is rejected, because it is the fastest way. After considering these cases the hierarchical

topology emerges from the lower level – Primer agents, through the middle one – Coordinators (also Saver might be considered here), to the highest level – Broker and Migration Coordinator. This theory is also proved by communication diagram, where the main communication thrust goes in those directions.

On the other hand, because there are connections between Coordinators and Primers, one way connection from Primers to remote Coordinator and the network between Migration Coordinator, Coordinators and Broker is almost complete (only one way connection between Broker and Migration Coordinator). The partial pattern of decentralized topology is realized here.

5 Results of the experiments

The ultimate goal is to find optimal configuration for two different environments: home and university. In order to do this there has been a set of experiments conducted. Other main purpose is to show, that migration helps to improve efficiency of task completion when a network is composed of computers, which are not homogeneous – they have different configuration or/and different computing power at the moment (they can be busy because of other programs running).

5.1 Meaningful Values for Parameters

When agents in the system are being created they get some parameters from the outside in order to control somehow the experiment process. Some of them were already mentioned, but without a detailed description.

Parameters for Coordinator agent:

- *Number of agents on a node.* This is the first parameter from Coordinator and it is given to speed up the agent creating network. When there are already Containers in the Agent Platform, they are ready to have the Coordinators created. Just after Coordinator is created it creates also Primers in the current location. This parameter is responsible how many of them will be initiated with the Coordinator. The default value is set to 10.
- *Valid time.* This is the parameter connected with the migration process and it was mentioned before. When after the migration phase a Coordinator gets all progress reports from Primers, it saves the state. It means the time of getting the last report and the sum of ranges of searching primes from. When a report request comes from Migration Coordinator, it sends progress report request again. If the difference between times of getting the last report from this phase and the saved is lower that valid time (also might be called trial minimum time) then a Coordinator estimates time based on the efficiency of a current node from the beginning of the experiment. Parameter is given in seconds and the default value is 1 second.

Parameters for Primer agent:

- *Migration type.* This argument is passed from Coordinator while creating a Primer. In the system there are two places when it comes to the migration decisions. The main role goes to the Migration Coordinator, because it decides where agents should go, but Primer agents decide how they go to the other environment. This parameter is a number and has values from 1 to 3. 1 corresponds to move migration type, 2 to copy and 3 to birth. There is a possibility

for agents on different nodes to have different types of migrations, but it is only as a start, because each Primer carries the migration type with it. Only when agent is born on another node it gets parameters from the Coordinator of the current node. In a standard situation Primers posses the type of migration on each node of the experiment. The default value is set for birth migration type.

- *Primes range search unit.* This is one of the most important parameter in the system (also called elementary primes range search). When a Primer is searching for prime numbers, it has to have a range of numbers to search from. This parameter is a number of numbers, which are checked at one time. After a Primer checks a unit of numbers, it generally checks if there are messages waiting to be handled. This parameter is also responsible for how often the save messages are generated. Every Primer agent after checking a range unit sends a message to the Saver agent in order to save the results (prime number). After primes are sent the list of primes held by the Primer agent is cleared. The default value is set to 150.

Parameters for Migration Coordinator agent:

- *Type of migration algorithm.* This parameter is a number and has minimal value of 1. Migration type algorithm mainly concerns how many agents are migrating during the one migration process. There are three possibilities. The value of 1 sets unlimited number of agents for migration.. The value of 2 sets algorithm for limited one. The main core of algorithm is of course the same, but algorithm stops when the number of proposed migrations reaches its limit. The value of 3 and higher sets mixed migration algorithm. First the unlimited migration is conducted and then after number of migration phases reaches its threshold, then limited migration is performed. The default value is set for 1.
- *Report time.* This parameter is given in a number of seconds, which have to elapse from the end of migration phase (when all Coordinators report local migration has finished) till the next report request sending by Migration Coordinator. The default value is set to 20 seconds.
- *Node threshold.* This parameter is given in a number of seconds. It is strictly connected with migration process. Coordinators are sending reports to Migration Coordinator - they send the estimated time till the job ends on the node and number of agents. When migration calculation starts, it needs all these values, but it does not include a node into calculation when its estimated time is lower than node threshold. Then this node is also omitted when calculating average active node time, sum of agents and sum of proposed agents. This parameter is given in order to avoid unnecessary migration. This migration is when a node has already finished its job and afterwards there are new agents arriving with some work to do. This parameter could also be used to anathematize nodes from migration earlier. The default value is set to 15 seconds.

5.2 Testbed

There are two environments, in which experiments were conducted. Both of them are reliable and experiment can be easily repeated.

The first one is a home network, which is composed of four computers with different configuration each:

Computer 1	*Intel Core Duo 1.86 GHz, 2 GB RAM, Windows Vista SP1*
Computer 2	*Intel 1.86 GHz, 448 MB RAM, Windows XP Prof. SP2*
Computer 3	*AMD Athlon 3200+ 2.2 GHz, 1 GB RAM, Windows XP Prof. SP2*
Computer 4	*AMD Athlon 1700+ 1.44 GHz, 256 RAM, Windows XP Prof. SP2*
Computer 5	*Intel 2,81 GHz, 448 MB RAM, Windows XP Prof. SP2*

Table 1: List of computers used in experiments

The network is Ethernet 100Mb/s. All computers are connected to the DI-604 Ethernet Broadband Routed, which is often used for home networking and small business networking. During the experiment there were only several problems with the network, but generally it is a stable environment.

The second environment is composed of computers at Wrocław University of Technology in laboratories for student practice in programming languages. The network is composed at least from 45 computers with the same parameters:

The speed of network is also 100Mb/s. All computers are also connected to a router. This is more reliable environment that the previous one. There were only several network problems, but they did not affect the experiment results

5.3 Performance Measurements

In order to evaluate the effectiveness of agent migration six groups of experiments were conducted: computer efficiency, maximum and minimum strategies, multi-agent migrations, parameters investigation, networked computers efficiency, and massive multi-agent migration. The results from first four groups are described in [Lupa, 08]. The results from the rest are as follows.

5.3.1 Networked Computer Efficiency Investigation

The experiment is set at university where all computers have the same power. The experiment started with one computer networked with another, which was only supposed to collect the prime numbers sent and save them to the file. This was done in order not to slow down one of the computers performing calculation with additional container and with file saving operations. When experiment continued, more computers were joining the network up to 46 of them.

Experiment parameters are as follows: 20 agents on each node, primes search range unit of 1000. The range of numbers to check for primes was from 1 billion to 1.004 billion.

Figure 8 presents the increase of efficiency with adding computers to the network. The shape of it was to be expected, but also for this number of computers there is point, form where the execution time starts to grow. Very important here are the parameters: 20 agents are calculating the prime numbers on each computer and the range of searching from primes for each agent equals 1000. This means every one thousand searched candidates for primes there is a break in order to check if there is a message for the calculating agent. Total range is from 1 billion to 1.004 billion. So after quick calculation there are at least 4000 so called “portions” to examine. If the

portion is lower than an agent checks message queue more often and its efficiency decreases, and so the experiment time rises.

Figure 8: Relationship between time and computers number solving the task

Computers	***Time***	***First reported***	***Diff.***	***Organisation messages***	***Saved messages***
1	*338,3*	*338.3*	*0*	*47*	*4000*
3	*113.5*	*112.4*	*1.1*	*133*	*4020*
5	*68.3*	*67.4*	*0.9*	*219*	*4000*
20	*17.6*	*16.9*	*0.7*	*864*	*4000*
45	*7.8*	*7.5*	*0.3*	*1939*	*4050*

Table 2: Partial results from networked prime number calculation

Table 2 presents the most important data from the networked prime number calculation. An interesting issue is the event of first computer reported. When prime numbers are being calculated, it is normal that some computers finish faster than others, even though they have the same parameters and almost the same number of operations to do (it depends on a value of the square root of a candidate number). The difference between the execution time and first reported computer is shown on the table below. At the beginning it is quite erratic but afterwards it stabilizes into descending trend. Moreover, it is important that the difference is very small, up to more than a second. The cause is the same configuration of all computers.

The number of organization messages in the experiment has a linear correlation with the number of computers in a network (on each computer there were 20 Primers). The number rises by 43 with each added computer and it starts from 4 messages, which are always necessary to make an agent network in the experiment.

Save messages are determined by the number of portions that agents get. Where there is low number of nodes in the experiment, this number does not go much higher than 4000, which is the result of dividing the range between low numbers of computers. When node number increases, it is almost impossible to distribute full 1000 ranges throughout all the agents, so the message number in some cases rises by 10%. But still this change does not affect the results (execution time) much. They seem to be in a steady descending trend.

There are several conclusions from this experiment:

- Finding the optimal number of computers for this experiment is a hard task. The optimum point is vague.
- The network in able to transfer all the data within designated time with almost no problems. There is not enough data than to make it slow down. Of course with different set of parameters that the default one, which seem to be close to optimal, this is possible.
- The ideal time is connected with the first computer reported in the network.
- Computers reporting the end of calculated primes are doing it within a short time from each other, which corresponds to their configuration.
- Number of organization messages is connected in a linear way with the number of computers.
- Number of saving messages is sensitive to the distribution of the main range of number to primes within.
- The difference between first machine reported and number of computer forms a descending trend.

5.3.2 Massive multi-agent migration

The experiment is conducted in order to test the implemented system using big number of computers. Because the university environment is homogeneous, there was a heterogeneity introduced by running more instances on one computer. Tests were conducted on 15 computers in which on five of them there was one JADE container running. On the next five there were two containers running and on the rest 3 containers. The number of containers equals 30. In this experiment the main goal is to find the optimal parameters for the described configuration. Without migration the experiment took 1000.5 seconds and the first computer reported after 317.6 seconds.

Experiment parameters are as follows: 10 agents on each node, primes search range unit of 100, report time set as 30 seconds, node threshold set for 15 seconds, the default migration type is birth. The range of numbers to check for primes was from 1 billion to 1.110 billion. The unlimited migration algorithm is used here.

Figure 9 shows that container times change with the changing report time parameter. The difference between here and experiments conducted at home is that one computer does not correspond to one container. Summing all number of containers – there are 30 of them here.

When the report time is low, the experiment time is higher and the first container reports quite early. When closing to the value of 30 seconds, the experiment time is lowering and the first container reports later. When the report time is between 25 and 35 seconds, the execution time is in a niche, where results are similar. In the whole chart the average container time is almost the same all the time. The reason here probably is the number of containers. The first container value has its maximum when the report time equals 30 seconds.

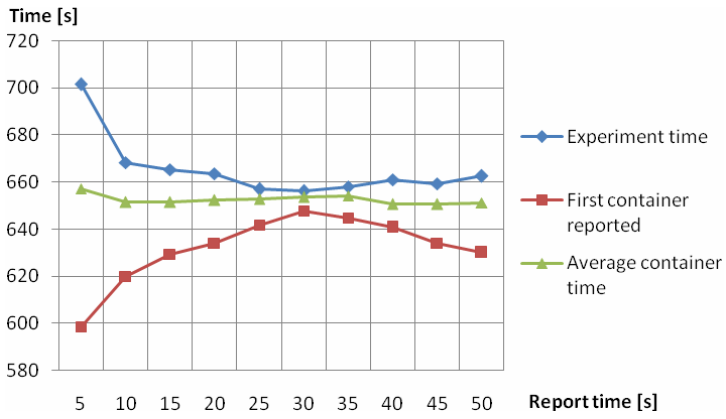

Figure 9: Relationship between container times and report time

As depicted on Figure 9, the optimal value here seems to be 30 seconds as a report time. Comparing to the execution time, it is quite a value – almost 5% of the whole time. It is also important here that the number of agents is low and the primes range search is small, which makes communication in the experiment quite fast. These values were chosen because of the large number of containers. We can clearly see that with low report time value the environment is very dynamic and with bigger values (like 50 seconds) it much less dynamic.

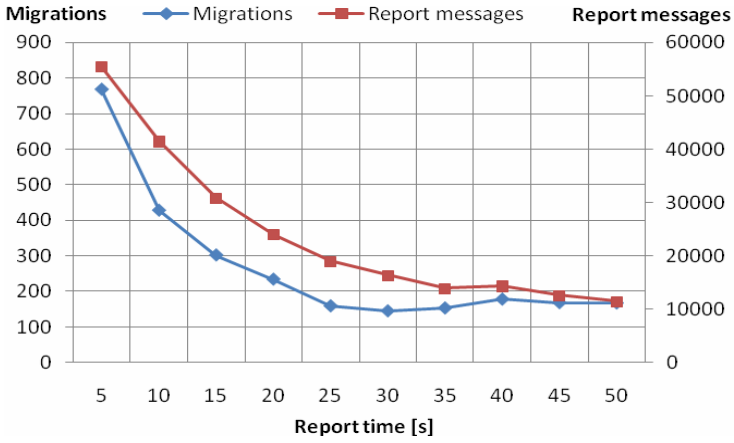

Figure 10: Relationship between report time, report messages and migrations

Figure 10 shows how many report messages are sent when the report time rises. Of course, when it does, the number of messages is lowering, but it is not the regular proportion, but counter proportion. This chart can also be interpreted as measure of systems dynamics. The higher the number of messages, the more agents and containers are participating in migration phases. After the report time crosses the value of 30 or 35 seconds, the number of messages is lowering in a slower pace. This

is on one hand the result of how many times agents are reporting to local Coordinators, but also how many containers and agents are participating in the migration process.

The number of migrations has its minimum around 25 to 35 seconds for the report time. Afterwards the migration numbers is a bit higher but it seems also quite stable and before these values it falls down rapidly. After the comparison of execution time and chart and migrations chart there is one additional conclusion – number of migrations is related to the execution time (the same minimum values).

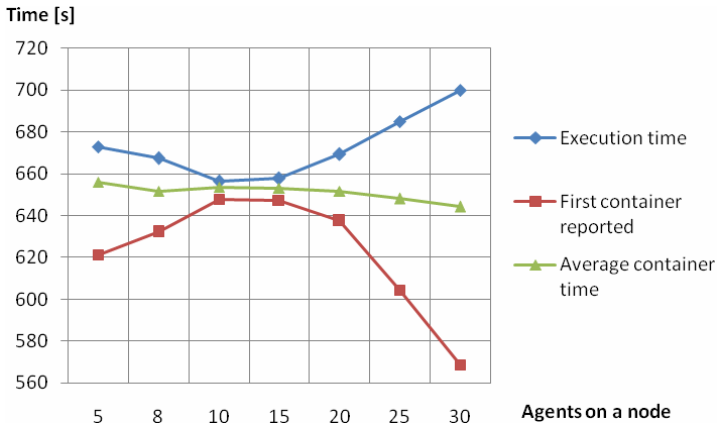

Figure 11: Relationship between agents on a node and container times

Figure 11 presents how the number of agents affects container times. Starting from execution time, it falls when the number of agents is nearing 10 from the number of 5 agents and rises starting from 15 agents on a node. The line of first container reported behaves in an exactly opposite manner and average container time is almost a constant value. The slight descending trend starts from 15 agents on a node.

The results are explainable. When there are only few agents, they cannot cover differences in computing power, so some containers finish earlier and when there are too many agents, report communication is more time-consuming and migration takes more time and so the system is not able to react quickly. This may be a problem when some agents are starting to finish their job and number of active agents in the system falls down. In that kind of situation containers reports more distributed through the time; migration is not able to cover those differences.

The most important conclusion from this chart is that the optimal number of agents stating on a node is between 10 and 15. For these two values the proximity of execution time, first container reported and average reported time is very high. That means the migration is able to cover computer efficiency differences quite well (difference between execution time and first container reported is less than 2%).

As shown in Figure 12, starting from the migrations number, it seems to be comparable in the range of agents from 5 to 10 and then it starts to rise rapidly. Then

it rises rapidly when the number of starting agents on a node crosses 15. The relationship between report messages and starting agents on a node is almost linear.

The number of migrations at the beginning of a chart, which is comparable, may be dependent on the minimum number of migrations during the experiment that makes the efficiency of a container close to optimal. As it is also depicted, the number of report messages does not correspond to the migration number in any way, beside both are generally rising when the number of agents on a nodes increases.

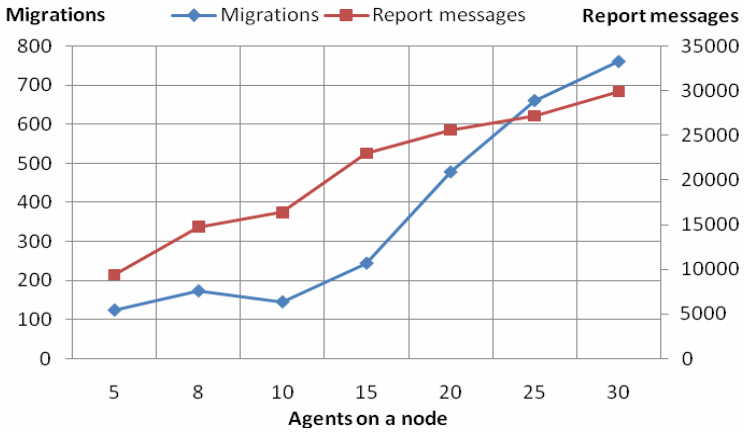

Figure 12: Relationship between agents on a node, migrations and report messages

After experiments with primes range search unit and computer times the conclusion was that with lowering the search range unit computers spend more time on checking if they have a message pending instead of calculating. The Figure 13 shows that again, but the shape is different.

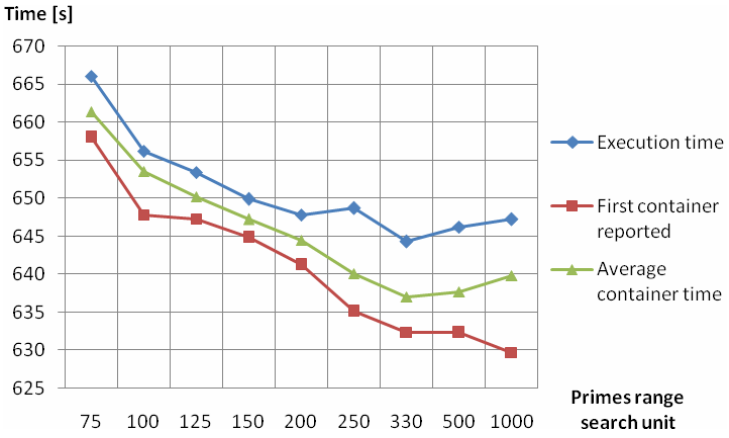

Figure 13: Relationship between primes range search unit and container times

There is a constantly descending trend of container times, but the difference between first container reported and execution time is getting bigger with increase of primes range search unit. Moreover, with the higher values of search range the execution time stabilizes between 645 seconds and 650 seconds, but the first container reports earlier. It is connected with the communication time during the experiment (the higher search range, the longer communication between agents). The close proximity of container times at the beginning is also a consequence of this relationship. The average container time seems to be in almost all cases in the middle between first container time and last container time (execution time).

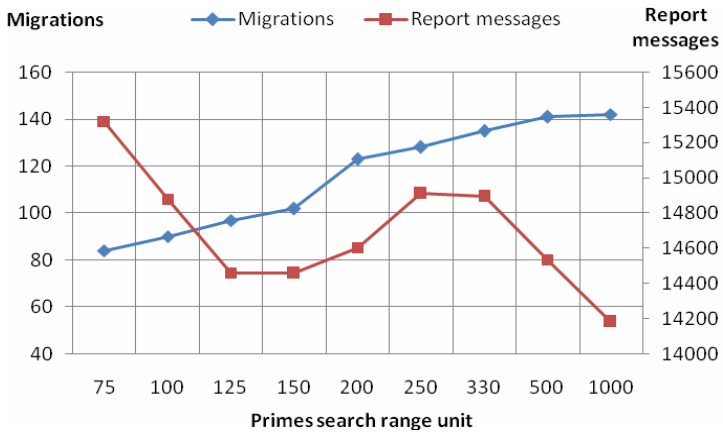

Figure 14: Relationship between migrations, report messages and primes range unit

Figure 14 shows that report messages and not related to migrations when the primes range search unit is changing. The migration number has almost linear relationship with the search range. The value change between the lowest and the highest values almost doubles.

The difference in report messages for the lowest value of primes range search unit and the highest one is around 8% (comparing the difference to the maximum value). The conclusion here is that the relationship is almost constant. The shape of it is more chaotic, because of the formed local minimum and maximum (beside values for at the beginning and at the end).

Figure 15 shows a comparison of migration types. For each value types (execution time, average container time, first container time) the differences are lower that 1%. An important factor here is that the experiment was conducted for primes range search unit value set as 150. Even though the conclusion here is that migration type does not have an impact on execution time. These results are more influenced by other parameters, like report time, primes range search unit or number of agents starting on a node.

There are several conclusions from this experiment:

- The optimal report time value is around 30 seconds. The minimal report time value, which is comparable with the optimal solution, is 10 seconds. 5 second time makes environment too much dynamic.

- Migrations and report messages have a connection between themselves when the report time is changing. The value of report time for migrations and report messages is comparable with the optimal value starting from the report time set for 20 or 25 seconds.
- The optimal number of starting agents on a node is between 10 and 15. Lower values make the environment too static and higher make it too dynamic.
- The lower agent number, the lower migration number.
- The optimal primes range search unit starts after values of 150, 200. It is also not clear because of relationship with containers time.
- Migration type does not influence the execution time.

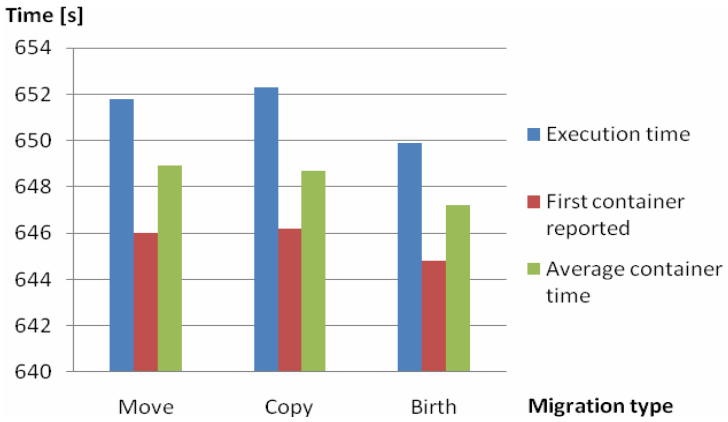

Figure 15: Relationship between computer time and migration type

6 Discussion and Conclusion

We spent countless hours making multiple runs in order to assure ourselves that the results were due to inherent randomness and not model errors. Table 3 provides the best values from these runs. Note that the agent migration increases the efficiency in task execution.

Parameter	***Heterogeneous computers***	***Homogeneous computers***
Profit in execution time	*58%*	*35.4%*
Primes range search unit	*150*	*150-200*
Report time	*20s*	*30s*
Agents on a node	*5-20*	*10-15*

Table 3: Optimal configurations for heterogeneous and homogeneous computers

Moreover, we can combine the results of all of above cases and draw the following conclusions. The more agents in the system, the more dynamic the environment is and also the more migrations take place. The most important parameter in the system is primes range search unit, because it is able to balance the calculations efficiency and the communication velocity. The optimal configuration is when all computers finish their tasks in time close to each other. The closer the migration phases are in time, the more migrations there are in the system. To make the system more stable migration phases have to be distant in time. This also impacts the number of messages in the system – the more stable the system is the lower communication costs. The lower number of migrations the shorter the execution time assuming there is enough agents in the system, that are able to cover differences in computer efficiency (the optimal number of agents seems to be around 10 or 15).

Investigations on migration types have given a conclusion that birth migration type is the more efficient one. It is also the simplest migration (fewer operations have to be done). In move or migration type operations like serialization and class loading have to be conducted, which slows down their efficiency. However, it is very important to mention that migration type does not have an impact on execution time. Other parameters responsible especially for load balancing are more crucial to efficiency of the distributed computing.

When there are migrations in the system, there is a possibility of cycle migration phenomenon. This has a negative impact on nodes efficiency and it was proven using full experiments report. The cause of this lies in the accuracy of time till end estimation for a node and the number of agents. There are two ways of limiting it: low number of agents or limited migration, but there is always a danger that after a sudden event in the system (like one node efficiency collapse) it could not handle changes in a short time (slightly higher execution time).

When the number of agents in the system is small, a limited migration could function more efficiently. The overall conclusion here is that within a dynamic environment the key is to find a balance between covering differences in computer efficiency and unexpected events (the more agents, the more accurate it is) and limiting migrations and migration cycles (the less agents the better).

The upgrades to AM framework could be connected with two key points: algorithm for migration and time estimation. More advanced algorithm for migration calculation could be more concentrated on avoiding migration cycles (something like limited migration) with parameters regarding how the node was handling the task previously. The time till end estimation could be more influenced by historical efficiency based on assumption that it does not change so often.

Finally the architecture of this system enables to introduce more task types. The prime numbers calculation is a simple one, but it also possess features of a distributed problem. With different problem agent migration types could reveal more differences if an agent would possess more data with itself. Moreover if a task is knowledge-intensive this knowledge could spread out and be exchange in the network. In summary, this work and system project opens new horizons in investigating code and data propagation in a multi-agent system.

Acknowledgements

The authors would like to express their gratitude to M. Nowostawski from the University of Otago and P. Luczycki from IBM, for their useful comments and suggestions to a draft version of this paper.

References

[Bellifemine, 08] Bellifemine, F., Caire, G.: A. Poggi, G. Rimassa, JADE: A software framework for developing multi-agent applications. Lessons learned, Information and Software Technology, 50, 2008, 10-21.

[Bernon, 06] Bernon, C., Chevrier, V., Hilaire, V., Marrow, P.: Applications of Self-Organising Multi-Agent Systems: An Initial Framework for Comparison, Informatica, 30, 2006, 73-82.

[Cao, 05] Cao, J., Spooner, D. P., Jarvis, S. A., Nudd, G. R.: Grid load balancing using intelligent agents, Future Generation Computer Systems, 21(1), 2005, 135-149.

[Centola, 06] Centola, D., Eguiluz, V.M., Macy, M.W.: Cascade dynamics of complex propagation, Physica A, 374, 2007, 449-456.

[Chakravarti, 05] Chakravarti, A.J., Baumgartner, G., Lauria, M.: The Organic Grid: Self-Organizing Computation on a Peer-to-Peer Network, IEEE Transactions on Systems, Man, and Cybernetics—Part A: Systems and Humans, 35(3), 2005, 373-384.

[Chmiel, 05] Chmiel, M., Gawinecki, M., Kaczmarek, P., Szymczak, M., Paprzycki, M.: Efficiency of JADE agent platform, Scientific Programming, 13, 2005, 1-14.

[DZhang, 08] Zhang, D., Simoff, S., Aciar, S., Debenham, J.: A multi agent recommender system that utilises consumer reviews in its recommendations, Int. J. Intelligent Information and Database Systems, 2(1), 2008, 69-81.

[Fatima, 01] Fatima, S., Wooldridge, M.: Adaptive Task and Resource Allocation in Multi-Agent Systems, In Proc. of the 5th International Conference on Autonomous Agents 2001, 537-544.

[Hmida, 08] Hmida, F.B., Chaari, W.L., Tagina, M.: Performance Evaluation of Multiagent Systems: Communication Criterion, In Proc. KES-AMSTA 2008, LNAI 4953, 2008, 773-782.

[Huang, 08] Huang, C.Y., Cheng, C.Y., Sun, C.T.: Resource Limitations, Transmission Costs and Critical Thresholds in Scale-Free Networks, In Proc. KES-AMSTA 2008, LNAI 4953, 2008, 485-494.

[JADE, 08] Jade - Java Agent DEvelopment Framework, available from: http://jade.tilab.com/ [cited 2008 September 30].

[Jiang, 05] Jiang, Y.C., Jiang, J.C.: A multi-agent coordination model for the variation of underlying network topology, Expert Systems with Applications, 29, 2005, 372-382.

[Jurasovic, 06] Jurasovic, K., Jezic, G., Kusek, M.: A Performance Analysis of Multi-Agent Systems, International Transactions on Systems Science and Applications, 1(4), 2006, 335-342.

[Kazienko, 07] Kazienko, P., Musiał, K.: On utilising social networks to discover representatives of human communities, Int. J. Intelligent Information and Database Systems, 1(3/4), 2007, 293-310.

[Korzeniowski, 08] Korzeniowski, Ł., Król, D.: Instant messaging data processing of autonomous mobile systems, In Knowledge processing and reasoning for information society, Exit Press, 2008, 83-99.

[Krivokapic, 00] Krivokapic, N., Kemper, M. A.: Migrating Autonomous Objects in a WAN Environment, Journal of Intelligent Information Systems, 15, 2000, 221-251.

[Krol, 08] Król, D., Zelmozer, M.: Structural Performance Evaluation of Multi-Agent Systems, Journal of Universal Computer Science, 14(7), 2008, 1154-1178.

[Krol, 09] Król, D., Kukla, G.: Quantitative Analysis of the Error Propagation Phenomenon in Distributed Information Systems, In Proc. of the 1st Asian Conference on Intelligent Information and Database Systems, IEEE Computer Society, 2009 (in press).

[Levis, 04] Levis, P., Patel, N., Culler, D., Shenker, S.: Trickle: A selfregulating algorithm for code propagation and maintenance in wireless sensor networks, Technical Report, University of California at Berkely, 2004.

[Lupa, 08] Król, D., Lupa, A.: Code and Data Propagation on a PC's Multi-Agent System, In New Trends in Multimedia and Network Information Systems, IOS Press 2008, 259-275.

[Mitschang, 03] Mitschang, B.: Data propagation as an enabling technology for collaboration and cooperative information systems, Computers in Industry, 52, 2003, 59-69.

[Navarro, 05] Navarro, G., Ortega-Ruiz, J.A., Ametller, J., Robles, S.: Distributed Authorization Framework for Mobile Agents, In Proc. MATA 2005, LNCS 3744, 2005, 127-136.

[Ni, 08] Ni, J., Luo, D., Ou, Y., Luo, C.: Agent-based evolutionary optimisation of trading strategies, Int. J. Intelligent Information and Database Systems, 2(1), 2008, 25-48.

[Repantis, 04] Repantis, T.: Adaptive Data Propagation in Peer-to-Peer Systems, Course Project Report for CS253 - Distributed Systems, University of California, Riverside, 2004.

[Tan, 03] Tan, K., Zhang, Q., Zhu, W.: Shortest Path Routing in Partially Connected Ad Hoc Networks, In Proc. IEEE Globecom 2003, 2, 2003, 1038-1042.

[Wedde, 06] Wedde, H.F., Farooq, M.: A comprehensive review of nature inspired routing algorithms for fixed telecommunication networks, Journal of Systems Architecture, 52, 2006, 461-484.

[Zhang, 08] Zhang, H.L., Leung, H.C., Raikundalia, G.K.:Topological analysis of AOCD-based agent networks and experimental results, Journal of Computer and System Sciences, 74, 2008, 255-278.