ISSN 1392–124X (print), ISSN 2335–884X (online) INFORMATION TECHNOLOGY AND CONTROL, 2014, T. 43, Nr. 1

Agent-Based Distributed Computing for Dynamic Networks

Dejan Mitrović[1], Mirjana Ivanović[1], Zoltan Geler[2]

[1]*Department of Mathematics and Informatics, Faculty of Sciences,*
University of Novi Sad, Serbia
e-mail: dejan@dmi.uns.ac.rs, mirag@dmi.uns.ac.rs

[2]*Faculty of Philosophy, University of Novi Sad, Serbia*
e-mail: gellerz@gmail.com

crossref http://dx.doi.org/10.5755/j01.itc.43.1.4588

Abstract. Dynamic networks of personal computers are characterized by sudden, unexpected failures of computational nodes, as well as the addition of new computers at any given moment. This paper presents a new agent-oriented system for distributed computing, named *ADiS*, specifically designed for operating in these kinds of environments. The system relies on the agent technology in order to adapt to dynamic changes in a computational network. Originally envisioned as a distributed system for distance matrix calculations, *ADiS* has recently been redesigned as a general-purpose, extensible architecture for arbitrary computing in a distributed environment. Experimental results demonstrate that the usage of *ADiS* has a significant positive impact onto the execution time of lengthy processes.

Keywords: software agents; distributed computing; dynamic networks; time-series analysis.

1. Introduction

A *time-series* is a chronologically ordered data sequence, recording states of an observed phenomenon over a period of time. Time-series data arise from and are used in many different fields of applications including finance, medicine, and various domains of science. The increasing need for applying time-series in order to solve various problems led to an explosive growth of interest in different research areas, such as classification, clustering, prediction, anomaly detection, indexing, and pattern discovery [7, 10, 14, 18]. All of these tasks rely on determining the similarity between the studied time-series. The chosen similarity measure needs to reflect the underlying (dis)similarity of the data represented by the time-series [8, 14].

Time-series analysis often requires a re-use of similarity values between the same set of series. In case of long time-series and/or similarity measures, it is very helpful to save the similarity values between time-series, and thus speed up the experiments. Similarities between the time-series of a dataset are kept in the form of a *distance matrix* where the element at (i, j) contains the distance between the i-th and the j-th time-series from the set.

Calculating the distance matrix itself is often a very time-consuming process. Our Department has several computer centers which could be used to distribute distance matrix calculations, and thus speed up the process significantly. However, the computers are being used for practical exercises and assignments in a number of computer science courses, and are otherwise freely available to students, for their personal use. A computer that is performing distance matrix calculations should therefore stop and free its resources as soon as occupied by a student.

In this paper, a novel, agent-based system for distributed computing is proposed. One of the main motivations for developing this new system, named *Agent-based Distributed computing System* (*ADiS*), was to speed-up the lengthy distance matrix calculation processes using an existing hardware infrastructure of personal computers. However, the system is also designed as *extensible* and is not limited to distance matrix calculations; new plug-ins can be added dynamically to support any form of distributed computations.

ADiS primary targets institutions and users who cannot afford high performance grid systems. It brings efficient load-balancing and job-distribution architecture to environments equipped with (possibly low-end) personal computers.

One of the key features of *ADiS* is adaptivity to dynamic changes in a computational network. Dynamic changes assume that existing computers

might fail and disappear suddenly, while new computers might be added at any time. *ADiS* relies on the concepts of the agent technology to detect these changes in a timely manner, and adapt its functioning accordingly.

The original idea for *ADiS* was presented in [22], while the core algorithms were developed earlier [20, 21], with the aim of adding fault-tolerance to our multi-agent framework *XJAF* [23]. The system has since been transformed into a general-purpose solution. Its algorithms have been optimized in order to reduce the overall network traffic and improve the process of job sharing, while the whole architecture has been experimentally validated, as shown later.

The rest of the paper is organized as follows. Section 2 presents the work related to agent-based distributed computing and dynamic network management. Section 3 outlines the architecture of *ADiS* and its core components in greater details. Experimental validation of the proposed system is given in Section 4. Finally, general conclusions and future research directions are presented in Section 5.

2. Related work

For most of its functionality, *ADiS* employs so-called *system-level* agents [25]. System-level agents bring high flexibility to the system, because new functionalities can be easily added in form of new agents. When used for network management, system-level agents have additional important benefits, such as dynamic network building and maintenance, efficient discovery of faulty network nodes, and remote maintenance features [12, 20, 21, 33].

In *ADiS*, the heartbeat connection is a core tool for propagating information across the network. Distributed *ADiS* nodes are actually organized into a regular graph, and the applied process of information spreading is based on the concept of *graph infection* [9, 11, 34]. A nice property of graph infection algorithms is that the data propagation time boundaries (both minimum and maximum) can be determined mathematically. This means that the efficiency of the data propagation process implemented in *ADiS* can be easily evaluated. An additional feature specific to *ADiS* is the way in which the network is established and used, with the goal of minimizing the overall network traffic.

A large part of scientific research related to agent-based load-balancing techniques is focused on grid computing; see e.g. [3, 6, 26, 31]. Instead, *ADiS* is well-suited for dynamic environments comprised of heterogeneous, often low- end personal computers. It brings efficient distributed computing to users who cannot afford high-performance grid systems.

Agent mobility has often been exploited as the main tool for load-distribution. The *Ripple load balancing* technique presented in [32] is based on a *credit value* assigned to each computer in a network. Any imbalance in the credit values dispatches mobile agents, which transfer computational tasks from computers with higher to computers with lower values, trying to achieve the initial balance. In [19], a set of divisible tasks is assigned to a network of computational nodes. A set of mobile agents is then dispatched, each carrying a single task in search for the best possible node. Therefore, the proposed load-balancing algorithm is focused on finding an optimum dispersion of agents. Although this algorithm is very well analyzed and shown to always maximize the use of available resources, it is limited by two requirements: each task has to be divisible into equally-sized sub-tasks, and the computation grid needs to be (performance-wise) homogeneous.

Instead of mobile, *ADiS* relies on stationary agents, in order to avoid the network and computational overhead often associated with agent mobility. It also makes no presumptions on the size of (sub-)tasks and operates in truly heterogeneous environments. But, more importantly, none of these systems deals with the discovery of failed and newly available computers to the extent implemented in *ADiS*.

Correct ordering of events is a key requirement of any distributed system. The use of logical clocks for event ordering, i.e. *Lamport timestamps*, has initially been proposed in [17]. Vector clocks [13] build on the concept of *Lamport timestamps* by supporting an arbitrary partial order of events. For dynamic systems, an *Interval Tree Clocks* algorithm has been proposed [2]. It is a relatively complex, but practical mechanism, that allows a completely decentralized assignment of IDs for newly created processes, and is efficiently adapted to the number of processes in the system.

ADiS combines the simplicity of *Lamport* timestamps with the graph infection- based propagation algorithm. Because *ADiS* manages physical computers, it does not require an ID generation mechanism. Instead, it relies on the computer's *IP* or *MAC* address for identification. Global state of the system can be collected at any computer in the system, and it is kept in a correct state by the propagation algorithm. To deal with dynamic properties, *ADiS* introduces the concept of *persistent Lamport timestamps*. If a computer is rebooted, its logical clock needs to continue increasing from the last known value. This feature is crucial, because each change it the computer's state needs to be associated with a greater timestamp than the previous one. To simplify the development, the computer's (physical) clock is used to simulate persistence, but, in general, a persistent counter could easily be implemented.

The analysis of existing research reveals that the majority of systems are focused on load-balancing. According to our knowledge, *ADiS* is the only agent-based architecture that provides each of its computers with a correct overview of the overall network state, and does so in an efficient manner.

3. *ADiS* architecture

ADiS is a general-purpose software system for distributed computing, specifically designed for dynamic networks. A high-level overview of its building blocks is shown in Fig. 1. Core functionalities of the proposed system can be summarized as follows:

- Dynamic network management. *ADiS* incorporates a special type of a software agent, named *HBAgent* (*HeartBeat Agent*), in order to detect dynamic changes in a computational network, and maintain the correct overview of the network state. Because of these functionalities, *HBAgent* is one of the most important components of the system.
- Dynamic distribution of computational jobs. When a computer with sufficient processing resources is detected, running *ADiS* instances will be asked to share part(s) of their pending jobs and transfer them to the target computer. In this process, the central role is played by *JobManager*. The manager monitors the resource consumption in its host computer, as well as the availability of other computers in the network.
- Extensibility. *ADiS* can be extended with new forms of *computational agents* (*CAgent*), software agents that perform the actual computations on the host computer. A specialization of this agent named *DIMAGAgent*, designed for calculating distant matrices, is presented later in Section 4. Life-cycles of *CAgents* are controlled by *JobManager*.
- Remote administration. *ADiS* can be accessed remotely by system administrators, in order to, for example, inspect the execution progress of existing jobs, or to specify new jobs. It exposes *ADiSNode*, a top-level *Remote Method Invocation* (*RMI*) service [29] with a well-defined interface.

For its functioning, JobManager partly relies on the low-level *PDModule* component. *PDModule* is the only platform-dependent component of *ADiS*. It uses the operating system's *API* for resource monitoring, idle input detection, etc. When, for example, it detects that the host computer has been idle for a period of time, *PDModule* spawns a new *ADiS* instance, which then joins the existing computational network. Similarly, if it detects some (mouse or keyboard) input from the user, *PDModule* will instantly kill the running *ADiS* process in order to release the computer's resource as soon as possible.

A more detailed insight into the functioning of *ADiS* and its central components is given in the following sub-sections.

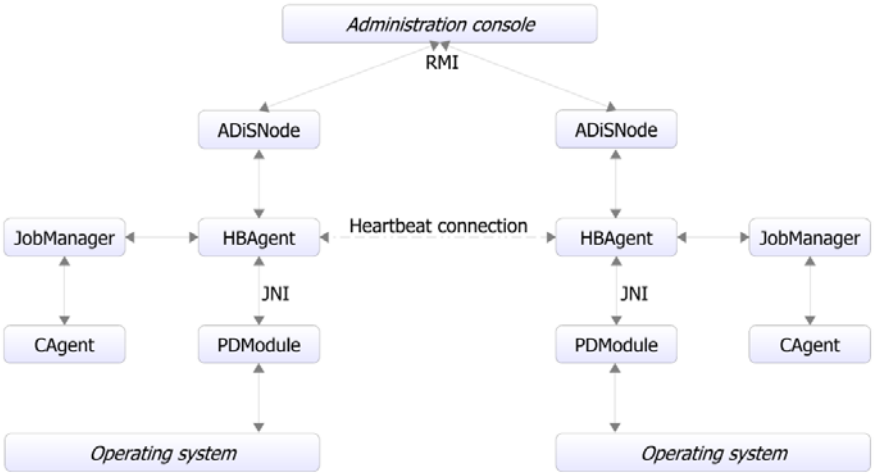

Figure 1. Architectural overview of *ADiS*

3.1. Managing dynamic networks

One of the core *ADiS* functionalities is the detection of changes in a computational network. Changes include the addition of new computers and unexpected unavailability of others. A correct and timely insight into the overall network state is crucial for the optimum job-sharing strategy. The dynamic network management is performed by a reactive, stationary agent named *HBAgent*. The agent is initially responsible for registering its host computer with an existing computational network. Then, it continuously monitors the overall network state, detecting any changes and signalling them to other agents.

There is a single instance of *HBAgent* in each computer that should become a part of the computational network. The instance if pre-configured with a list of *potential neighbors*, i.e. computers it should try to connect to during startup. All running *HBAgents* belong to a specially designed heartbeat connection. The connection is based on pings, signals that agents emit to each other in order to signify their presence and availability. Pings are emitted at regular time intervals; if an agent does not generate any pings within a certain time threshold, or if it fails to respond to an external ping, it is assumed to have become unavailable.

Each *HBAgent* keeps a list of every computer in the network, including its own host. An element of the

list is a record describing the agent's view of the corresponding computer's state. The state includes:

- Network address (also used as an identifier)
- Availability (i.e. *alive* or *dead*)
- List of assigned computational jobs
- An auto-generated timestamp in which the state was perceived

The heartbeat connections are maintained using the following algorithm. When emitting a ping to *HBAgent* H_j, *HBAgent* H_i serializes its list of all computers and includes it in the ping. Upon receiving the ping, H_j compares this information with its own list. During the comparison, it relies on timestamps for determining the correct state of a computer. If, for example, H_i thinks a computer is *dead*, while H_j thinks the computer is *alive*, the correct state is the one with a later timestamp. Any new information contained in the ping is incorporated in the H_j's list. In a response to the ping, H_j includes any new information H_i should incorporate in its own list. The response includes an updated state of at least one computer - the H_j's host. The updated state includes the latest timestamp at a minimum, and, optionally, a new list of assigned computational jobs. Listing 1 shows the implementation of the ping handler method, demonstrating the given algorithm.

Listing 1. Implementation of the ping handler

```
void onPing (NeighborList other,
NeighborList delta) {
   // check availability of system's
   resources , etc. updateMyState () ;
   // calculate the delta for each
   neighbor
   for (Neighbor node : neighbors) {
    int n = other.indexOf(node) ;
    // if not in his list , or mine has
    a greater timestamp...
    if ((n = -1)
     ||(node.getTimestamp()>
     other.get(n).getTimestamp()))delta
     .add(node);}
   // save his list locally
   // (operates similarly as the for-
   each loop above) neighbors . addAll (
   other ) ; }
```

In any multi-agent system [5], it is necessary to optimize the exchange of messages, since it often represents a performance bottleneck [1]. Therefore, the heartbeat connection is organized in special way. An agent first sorts its list of all computers according to their network addresses. Given that its position in the list is i, the agent establishes a direct heartbeat connection (i.e. exchanges pings) only with agents H_{i-1} and H_{i+1} (more precisely, it establishes a direct heartbeat connection with first alive agents H_{i-p} and $H_{i+q}, p, q > 0$).

If during the exchange of pings the agent H_i detects that, for example, its direct heartbeat neighbor H_{i+1} has become unavailable, it will update the corresponding record in its own list, setting the availability to dead and the new timestamp to the old timestamp plus an ϵ. Similarly, H_{i+2}, as the right-side heartbeat neighbor of H_{i+1} performs the same set of steps. Because it has been updated, the new state of H_{i+1} is included in pings signalled by both H_i and H_{i+2} and subsequently propagated to all computers. Simultaneously, H_i and H_{i+2} become new heartbeat neighbors.

HBAgent is based on a mobile *ConnectionAgent* originally proposed in [20, 21]. Whereas *Connection-Agent* was specialized for maintaining a fault-tolerant network of multi-agent systems, *HBAgent* is a more generalized solution, designed for general-purpose computational networks. Rather than mobile, *HBAgent* has been designed as a stationary agent, in order to reduce the network and computational overhead of agent mobility. Finally, *HBAgent* utilizes a new, custom serialization scheme and socket-based programming, which has a significant effect on reducing the network bandwidth required by pings, as discussed later.

3.2. Runtime job sharing

As noted earlier, one of the main design goals for *ADiS* is extensibility. New *CAgents* can be added to the system as needed, even at runtime, through a dynamically-loaded configuration file.

The system defines two base classes that need to be extended for each concrete computational agent:

1. *CAgent* - Represents all computational agents. The two of its most important methods are shown in Listing 2.
2. *JobDesc* - Description of a job that the computational agent supports. Additionally, each job has a global identifier (an *UUID*) automatically assigned to it.

When the heartbeat connection reveals that there is a remote computer with sufficient processing resources, a local *CAgent* is asked to share a part of its current job. It is up to the concrete *CAgent* implementation to determine how the job is actually divided into sub-jobs and shared. The shared part will be transferred to the remote computer, which may still refuse to accept it if, for example, it has accepted another job in the meantime.

Listing 2. Methods of the core JobSharingI interface for sharing jobs with distributed computational agents

```
public interface JobSharingI {
   // asks the agent to share (a part
   of) its current job
   JobDesc share () ;
   // informs the agent that the
   previously shared job has been
   // accepted for processing by
   another CAgent
   void jobAccepted (JobDesc jobDesc);}
```

If the remote *CAgent* accepts the job, the local *CAgent* will be informed through its *jobAccepted()* method. At the same time, the local *HBAgent* remembers (sub-)jobs that were delegated to the remote computer. This information is propagated through the heartbeat connection (initially, by the local and remote *HBAgents*) so that every *HBAgent* knows which jobs are processed in every other computer. In case of a computer failure, any *HBAgent* can take over its jobs (by convention, it is its right-side heartbeat neighbor).

There are two main ways of saving computational results. They can either be propagated through the heartbeat connection, just like any other information, or stored directly on a file server. The first approach is well-suited for a completely distributed system, and has the advantage of being more flexible (e.g. the computational results can be collected at any computer). However, it has a disadvantage of generating more network traffic. The second approach uses far less traffic, but is inflexible, in a sense that the file server must not fail.

ADiS supports both approaches. If a file server is specified in a configuration file, computational results will be sent directly there. Otherwise, they will be incorporated into pings of the heartbeat connection to be propagated across the computational network. The system defines a *RMI* interface that should be implemented by a file server.

3.3. Analysis of the heartbeat connection

Several steps have been taken to reduce the overall network traffic generated by pings. A custom (de)serialization scheme has been implemented to eliminate the metadata used by the Java *Serialization API*. Socket-based programming has been used instead of, for example, much simpler *RMI*, in order to, again, avoid the metadata overhead associated with each RMI call. However, the most important optimization factor lays in the organizational structure of the heartbeat connection.

As noted, each HBAgent sends pings to its heartbeat neighbors at *regular* time intervals, in order to signify its presence in the network. Besides the special organizational structure of the heartbeat connection, several additional steps have been taken in order to reduce the overall network traffic generated by pings.

A custom serialization scheme has been implemented, in place of the Java *Serialization API*. Using the new scheme, the size of an entire ping (which incorporates information about all neighbors) is $1 + 19 * n$ bytes, where $n > 1$ is the number of computers in the network. On the other hand, Java *Serialization API* requires $644 + 67 * n, n > 1$ bytes per ping. Fig. 2 shows the relation of ping sizes to the number of computers in the network, for both *ADiS* and Java serialization schemes. As it can be concluded, the new serialization scheme reduces the network traffic significantly.

Another very important factor of the heartbeat connection is the rate in which pings are generated (i.e. the heartbeat rate). The higher the rate, the faster *HBAgents* become aware of the changes in the network, but also more, possibly unnecessary, network traffic is generated. To help determine an opti¬mum heartbeat rate, a series of tests were performed.

Fig. 3 shows the amount of network traffic (per computer, per second) generated by different ping intervals (4, 8, 16, and 32 seconds) and for different network sizes (4, 8, and 16 computers). The given values were obtained by using *NetworkTrafficView*, a free network monitoring tool [24]. Therefore, the results include the real TCP packet sizes, and not just the amount of data a *HBAgent* reads from or writes to a socket. As expected, when the ping interval doubles (e.g. from every 8 to every 16 seconds), the network traffic is halved.

Next, the correlation of ping intervals and the time needed to distribute a piece of information across all computers in a network is analyzed. After the network has been established, an integer value is sent to a randomly selected *HBAgent*. From then on, the value is propagated only through the heartbeat connection. The experiment is finished once each *HBAgent* reports to have received the value.

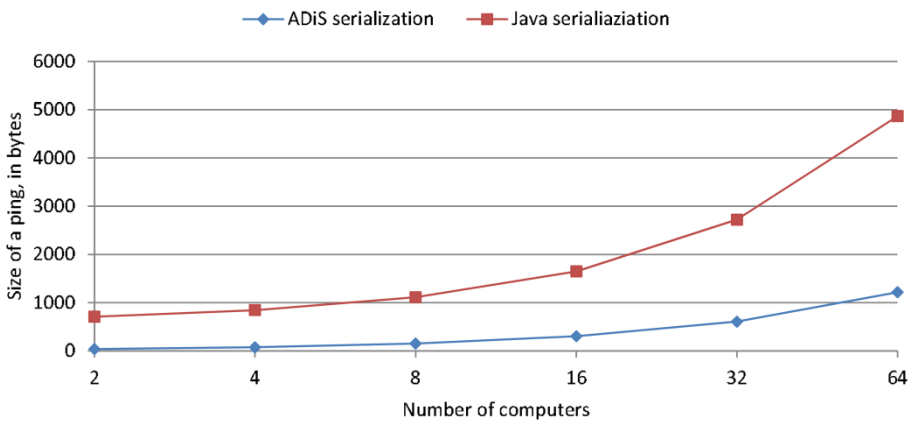

Figure 2. Relation of ping sizes to the number of computers in the network, for both custom serialization scheme used in *ADiS*, and Java Serialization API

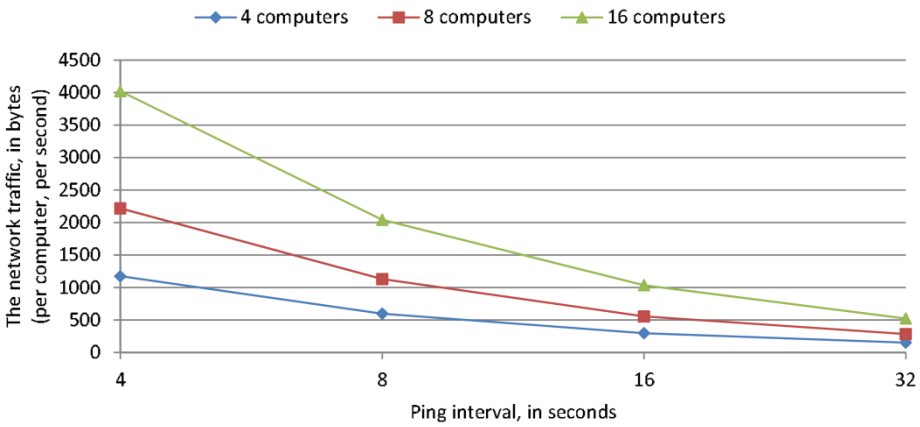

Figure 3. The amount of network traffic (per computer, per second) generated by different ping intervals and for different network sizes

As noted earlier in Section 2, this data propagation algorithm actually belongs to the class of *graphs infection* algorithms, which have been extensively studied. It has been shown in [34] that for a k-regular graph with n nodes, the upper bound for the number of pings needed to spread an information from one computer to all others is set to:

$$\lceil \log_{\lambda(k)} n \rceil + 3, \lambda(k) = \frac{k+\sqrt{k^2+4}}{2} \quad (1)$$

However, in the propagation algorithm used in [34], at ping p_t each even computer H_{2i} sends a message to H_{2i+1}, and at ping number p_{t+1}, each odd computer H_{2i+1} sends a message back to H_{2i}, $i \geq 0$. In the *HBAgent's* heartbeat connection, at each ping the computer H_{2i} sends a message to H_{2i+1} and then receives a message back from H_{2i+1}. This means that the *HBAgent's* heartbeat connection needs only half the number of pings required by the algorithm described in [34]. Given this analysis, and the fact that the *HBAgent's* heartbeat connection is a 2-regular graph, the upper bound for the number of pings needed to spread an information across the entire computational network becomes:

$$\left\lceil \frac{\log_{1+\sqrt{2}} n+3}{2} \right\rceil. \quad (2)$$

Fig. 4 outlines this upper bound, the worst possible values for the time needed to spread an information across the computational network for different ping intervals (4, 8, 16, and 32 seconds) and for different network sizes. Actual experimental results are shown in Fig. 5. They validate the upper bound for the ping, but also reveal that the average time is significantly shorter.

These two sets of information, along with the amount of network traffic shown in Fig. 3, can be used for determining the optimum heartbeat rate. More concretely, a ping interval in the range of 16 to 32 seconds results in good data propagation times, while generating low amount of network traffic even for a large number of computers.

4. Practical application of *ADiS*

ADiS has been practically employed in an automatic distribution of distance matrix calculations for time-series analysis. The main motivation, as described earlier, is to shorten the time needed to

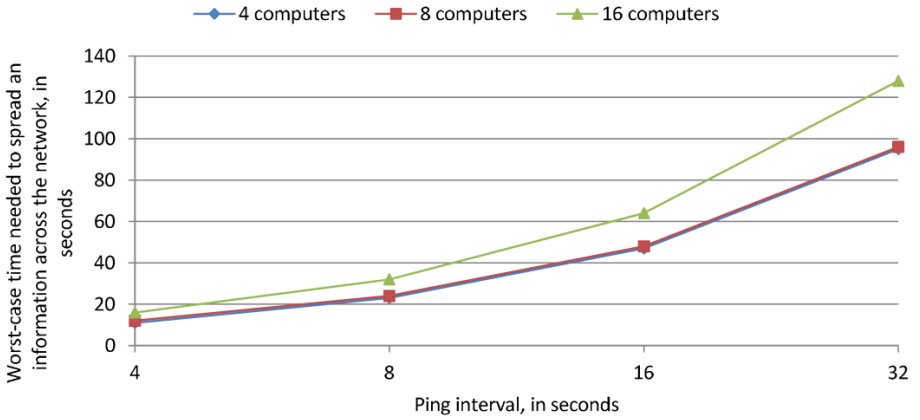

Figure 4. The upper bound for the time needed to spread an information across all computers in a network, for different ping intervals and network sizes

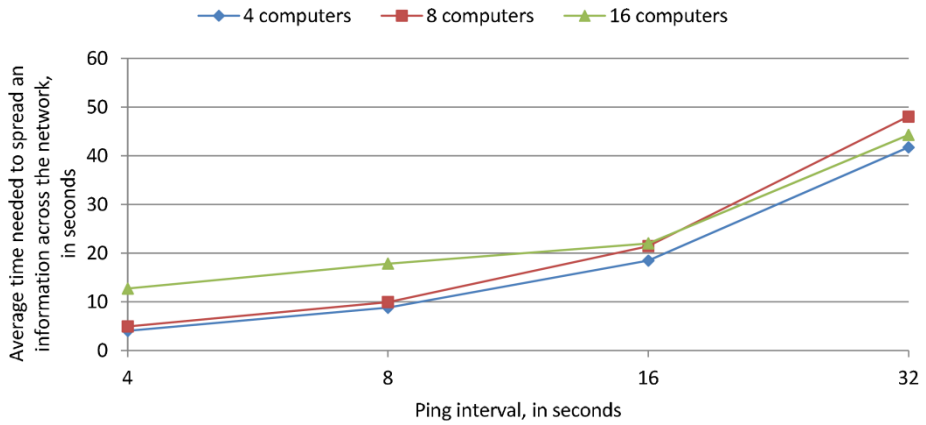

Figure 5. Experimentally evaluated times needed to spread an information across all computers in a network, for different ping intervals and network sizes

populate a distance matrix. The experiment presented in this section will demonstrate the runtime efficiency of *ADiS* in shortening the time of lengthy calculation processes.

4.1. DIMAGAgent

To enable distributed distance matrix calculations in *ADiS*, a new computational agent, named *DIMAGAgent*, has been developed. The agent is a simple wrapper around an existing *DIMAG* (*Distance Matrix Generator*). An important feature of the *DIMAG* component is its ability to split the problem of generating a matrix into a number of (approximately) equal parts, and later merge the resulting partial matrices into the final output. *DIMAG* generates the distance matrices using *Framework for Analysis and Prediction* (*FAP*), a free, platform-independent open-source library that encompasses several important algorithms for different temporal data-mining and time-series analysis tasks [16].

DIMAGAgent's job consists of one or more instructions, each of which contains the following information:

- Input dataset name
- Name of the similarity measure
- The total number of parts to split the distance matrix into
- Index of the part that needs to be calculated by this instruction
- Required matrix type, i.e. diagonal or full
- A similarity-dependent set of parameters

Therefore, each instruction is used to calculate a single part of the matrix. The job-sharing algorithm is executed as described earlier. When an available computer is detected and a *DIMAGAgent* receives the *share* request, it splits its own set of instructions into two distinct parts. One part is kept locally, while the other is marked as *shared* and sent to the remote computer. If the remote computer accepts these instructions, only then does the local *DIMAGAgent* exclude them from its job.

4.2. Experimental results and analysis

In the world of speech recognition, the *Dynamic Time Warping* (*DTW*) distance measure has long been a very well known technique. Two decades ago, it was introduced to the world of data mining in order to overcome some limi-tations of the *Euclidean* distance [4]. Since then, it has been successfully used for dealing with time-series in various fields, including medicine, engineering, bioinformatics, etc. [27].

DTW relies on dynamic programming and requires the comparison of each element of one time series with each element of the other one. This makes the calculation quite slow, and has some limitations regarding large datasets. One way to accelerate the computation is to narrow the search path in the matrix using global constraints. For example, the *Sakoe-Chiba band* [30] narrows the warping window around the diagonal of the matrix using a constant range r.

Distance matrices used in the presented experiment were based on the DTW distance measure, constrained by the *Sakoe-Chiba* band with the constant range r set to 10. All datasets used in the experiment were provided by *UCR Time Series Repository* [15], which includes the majority of all publicly available, labelled time-series datasets in the world. The actual datasets used include:

- 50words [28]
- FaceAll [35]
- Symbols
- Haptics

The main purpose of the experiment was to determine the efficiency of the heartbeat connection and job sharing algorithms implemented in *ADiS*. The system, along with *DIMAGAgent*, was deployed onto a number of low-end computers, all featuring the same hardware components: an *AMD Sempron 2800+* processor running at 1.61 *GHz* with 512 *MB* of *RAM*. The operating system used was *Microsoft Windows XP Professional* with *Service Pack 3*.

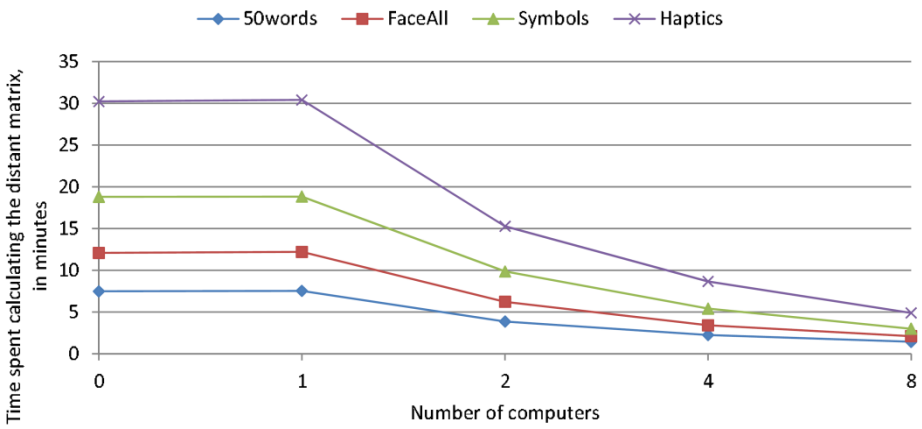

Figure 6. Results of the experimental evaluation of *ADiS* in distributing the distance matrix calculation process, for different datasets and varying number of computers

The heartbeat rate was set to 8 seconds. A separate computer was set up to act as a file server. Additionally, this computer hosted the *ADiS Remote Manager* application, used by the system administrator for sending new jobs to the running *ADiS* instances, as well as for monitoring the overall state of the distributed system.

Fig. 6 shows the experimental results. For each of the four datasets, it outlines the total time needed to compute the distance matrix by distributing it on 1, 2, 4, and 8 computers, respectively. To make the results more comparable, *DIMAG* was manually executed as a stand-alone application for each of the dataset, and the execution time was measured. In Fig. 6, these measurements are marked as "0 computers". Therefore, the difference between "0 computers" and "1 computer" is used to determine the overhead of using *ADiS*.

The overall results are very encouraging. They demonstrate that *ADiS* operates very well, and that each instance is able to efficiently detect available neighbors and share a part of its current job. Distance matrix calculation times converge uniformly as the number of computer increases, regardless of the dataset used. The results also demonstrate that there is a minimal overhead of using *ADiS*, i.e. that a single computer running *ADiS* operates as well as a stand-along *DIMAG* component.

Undoubtedly, the maximum number of computers used in the experiment is very small when compared to modern distributed (especially, grid) systems. However, the purpose of this experiment was mainly to test the runtime efficiency of *ADiS*, as well as to demonstrate the convergence of calculation times. In practice, the maximum number of computers in a network is not limited by any factor.

5. Conclusions and future work

This paper proposes a new agent-oriented system for distributed computing, named *ADiS*, specifically designed to operate in dynamic environments. Initially, *ADiS* has been envisioned as a system for distributing the process of distance matrix calculations in the scientific field of time-series analysis. However, it has since been extended to serve as a general-purpose architecture for distributed computing.

The two main characteristics of *ADiS* can be summarized as follows:

- Adaptivity to changes in dynamic networks. *ADiS* relies on the agent technology to observe changes in a computational network and adapt its behavior accordingly.
- Extensibility. The system has a well-defined plug-in architecture, where plug-ins are used to perform arbitrary calculations and rely on *ADiS* for job distribution.

Instead of many existing systems that target high-performance grid systems,

ADiS is aimed at users and institutions that are limited to networks of (possibly low-end) personal computers.

A lot of effort has been put into optimizing the overall network traffic used by the system itself. As demonstrated, the custom organization of the heartbeat connection and a new serialization scheme result in a significant reduction of the network bandwidth requirements.

The architecture has been experimentally validated as a framework for distributed distance matrix calculations. The experiments have shown that there is a significant positive impact of using *ADiS*: distance matrix calculation times converge uniformly as the number of computer increases, regardless of the dataset used.

Future research directions will be aimed at optimizing the network requirements of *ADiS* event further. The system will be extended with a possibility of handling failed connections between computers as well.

In the long run, *ADiS* needs to be extended with more advanced load-distribution and job-sharing

algorithms. This will be a non-trivial task, due to the use of plug-ins as custom computational entities.

Acknowledgments

This work was partially supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia, through project no. OI174023: "Intelligent techniques and their integration into wide-spectrum decision support."

References

[1] **J. M. Alberola, J. M. Such, V. Botti, A. Espinosa, A. Garcia-Fornes**. A scalable multiagent platform for large systems. *Computer Science and Information Systems*, January 2013, Vol. 10, No. 1, 51–77.

[2] **P. S. Almeida, C. Baquero, V. Fonte.** Interval tree clocks: a logical clock for dynamic systems. In: *Proceedings of the 12th International Conference on Principles of Distributed Systems, OPODIS '08*, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 259–274. http://dx.doi.org/10.1007/978-3-540-92221-6_18.

[3] **T. E. Athanaileas, N. D. Tselikas, G. V. Tsoulos, D. I. Kaklamani.** An agent-based framework for integrating mobility into grid services. In: *Proceedings of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications MOBILWARE '08, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels*, Belgium, 2007, pp. 31:1–31:6. http://dl.acm.org/citation.cfm?id=1361492.1361531.

[4] **D. Berndt, J. Clifford.** Using dynamic time warping to find patterns in time series. In: *AAAI-94 workshop on knowledge discovery in databases*, 1994, pp. 229–248.

[5] **C. Bădică, Z. Budimac, H. D. Burkhard, M. Ivanović.** Software agents: Languages, tools, platforms. *Computer Science and Information Systems*, 2011, Vol. 8, No. 2, 255–298.

[6] **J. Cao, D. P. Spooner, S. A. Jarvis, G. R. Nudd.** Grid load balancing using intelligent agents. *Future Gener. Comput. Syst.*, 2005, Vol. 21, No. 1, 135–149. http://dx.doi.org/10.1016/j.future.2004.09.032.

[7] **G. Das, D. Gunopulos.** Time series similarity and indexing. In: *Time series similarity and indexing, Lawrence Erlbaum Associates*, Inc., 2003, 279-304.

[8] **G. Das, D. Gunopulos, H. Mannila.** Finding similar time series. In: *J. Komorowski, J. Zytkow (eds.), Principles of Data Mining and Knowledge Discovery, Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 1997, Vol. 1263, 88–100. http://dx.doi.org/10.1007/3-540-63223-9_109.

[9] **T. Dimitriou, S. Nikoletseas, P. Spirakis.** Analysis of the information propagation time among mobile hosts. In: *Nikolaidis, I., Barbeau, M., Kranakis, E. (eds.), Ad-Hoc, Mobile, and Wireless Networks, Lecture Notes in Computer Science*, Springer Berlin/Heidelberg, 2004, Vol. 3158, 630–630.

[10] **H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, E. Keogh.** Querying and mining of time series data: experimental comparison of representations and distance measures. In: *Proc. VLDB Endow.*, August 2008, Vol. 1, No. 2, pp. 1542–1552. http://dx.doi.org/10.1145/1454159.1454226.

[11] **M. Draief, A. Ganesh**. A random walk model for infection on graphs: spread of epidemics & rumours with mobile agents. *Discrete Event Dynamic Systems*, March 2011, Vol. 21, No. 1, 41–61. http://dx.doi.org/10.1007/s10626-010-0092-5.

[12] **T. C. Du, E. Y. Li, A. P. Chang.** Mobile agents in distributed network management. *Communications of the ACM*, 2003, Vol. 7, Nr. 46, 127–132.

[13] **C. J. Fidge.** Timestamps in message-passing systems that preserve the partial ordering. In: *Proceedings of the 11th Australian Computer Science Conference*, 1988, Vol. 10, No. 1, 5666. http://sky.scitech.qut.edu.au/~fidgec/Publications/fidge88a.pdf.

[14] **J. Han, M. Kamber.** Data mining: concepts and techniques. *Morgan Kaufmann Publishers*, 2005.

[15] **E. Keigh, X. Xi, L. Wei, C. Ratanamahatana.** The UCR time series classification/clustering page. 2006. www.cs.ucr.edu/~eamonn/time_series_data.

[16] **V. Kurbalija, M. Radovanović, Z. Geler, M. Ivanović.** A framework for time-series analysis. In: *Proceedings of the 14th international conference on Artificial intelligence: methodology, systems, and applications, AIMSA'10*, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 42–51. http://dl.acm.org/citation.cfm?id=1885962.1885968.

[17] **L. Lamport.** Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, July 1978, Vol. 21, No. 7, 558–565. http://doi.acm.org/10.1145/359545.359563.

[18] **S. Laxman, P. Sastry.** A survey of temporal data mining. *Sadhana 31*, 2006, 173–198. http://dx.doi.org/10.1007/BF02719780.

[19] **J. Liu, X. Jin, Y. Wang.** Agent-based load balancing on homogeneous minigrids: macroscopic modeling and characterization. *IEEE Transactions on parallel and distributed systems*, 2005, Vol. 16, No. 7, 586–598.

[20] **D. Mitrović, Z. Budimac, M. Ivanović, M. Vidaković.** Improving fault-tolerance of distributed multi-agent systems with mobile network-management agents. In: *Proceedings of the International Multiconference on Computer Science and Information Technology*, October 2010, Vol. 5, pp. 217–222.

[21] **D. Mitrović, Z. Budimac, M. Ivanović, M. Vidaković.** Agent-based approaches to managing fault-tolerant networks of distributed multi-agent systems. *Multiagent and Grid Systems*, December 2011, Vol. 7, No. 6, 203–218.

[22] **D. Mitrović, Z. Geler, M. Ivanović.** Distributed distance matrix generator based on agents. In: *Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics WIMS 2012*, p. Article 40, ACM, New York, NY, USA, 2012.

[23] **D. Mitrović, M. Ivanović, Z. Budimac, M. Vidaković.** Supporting heterogeneous agent mobility with ALAS. Computer Science and Information Systems, 2012, Vol. 9, No. 3, 1203–1229.

[24] http://www.nirsoft.net/utils/network_traffic_view.html, 2012.

[25] **T. R. Payne, M. Paolucci, R. Singh, K. Sycara.** Facilitating message exchange through middle agents. In: *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part* 2., 2002, pp. 561–562.

[26] **M. Pipattanasomporn, H. Feroze, S. Rahman.** Multi-agent systems in a distributed smart grid: design and implementation. In: *Proc. IEEE PES 2009 Power Systems Conference and Exposition, PSCE'09*, 2009, pp. 1–8.

[27] **C. A. Ratanamahatana, E. Keogh.** Three myths about dynamic time warping. In: *Proceedings of SIAM International Conference on Data Mining (SDM '05)*, 2005, pp. 506–510.

[28] **T. Rath, R. Manmatha.** Word image matching using dynamic time warping. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2003, Vol. 2, 521–527.

[29] http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html, 2012.

[30] **H. Sakoe, S. Chiba.** Dynamic programming algorithm optimization for spoken word recognition. In: *Readings in speech recognition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA*, 1990, pp. 159–165. http://dl.acm.org/citation.cfm?id=108235.108244.

[31] **M. A. Salehi, H. Deldari.** A novel load balancing method in an agent-based grid. In: *IEEE International Conference on Computing and Informatics*, ICOCI 2006, pp. 1−6.

[32] **P. K. Sinha, S. R. Dhore.** Multi-agent optimized load balancing using spanning tree for mobile services. *International Journal of Computer Applications*, 2010, Vol. 1, No. 6, 35–42.

[33] **R. Stephan, P. Ray, N. Paramesh.** Network management platform based on mobile agents. *International Journal of Network Management*, 2004, Vol. 14, No. 1, 59-73.

[34] **V. S. Sunderam, P. Winkler.** Fast information sharing in a complete network. Discrete Appl. Math., February 1993, Vol. 42, No. 1, 75-86, http://dx.doi.org/10.1016/0166-218X(93)90180-V

[35] **L. Wei, E. Keogh.** Semi-supervised time series classification. In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining KDD'06*, ACM, New York, NY, 2006, pp. 748-753.

Received June 2013.