Fault Tolerance through Interaction and Mutual Cooperation in Hierarchical Multi-Agent Systems

Rade Stanković[1] and Maja Štula[2]
[1]*Communications, Media and Technology (CMT), Siemens d.d., Put Brodarice 6, 21000, Split, Croatia*
[2]*Department of Electronics, Faculty of Electrical Engineering,*
Mechanical Engineering and Naval Architecture in Split (FESB), R. Boškovića 32, 21000, Split, Croatia

Keywords: Fault Tolerance, Failure Handling, Multi-Agent Systems, Intelligent Agents, Agent Hierarchy.

Abstract: Multi-Agent Systems (MASs) are well suited for development of complex, distributed systems. In its essence MAS is a distributed system that consists of multiple agents working together to solve common problems. Failure handling is an important property of large scale MAS because the failure rate grows with both the number of the hosts, deployed agents and the duration of agent's task execution. Numerous approaches have been introduced to deal with some aspects of the failure handling. However, absence of centralized control and large number of individual intelligent components makes it difficult to detect and to treat errors. Risk of uncontrollable fault propagation is high and can seriously impact the performance of the system. Although existing research has been extensive, it still needs to attend the MAS failure handling problem in all its aspects, which makes this topic very interesting. We propose a concept of agent interaction that enables any hierarchical MAS to become fault tolerant, regardless of the used agent framework.

1 INTRODUCTION

Employment of intelligent agents in distributed systems incorporates concepts from both Artificial Intelligence (AI) and Comuter Science (CS). From the Computer Science perspective agent can be described as a peace of code, that when executed has data and state (Tanenbaum and Steen, 2002). From the Artificial Intelligence point of view, agents are defined as executable software entities with varying degrees of intelligence, autonomy and ability to communicate to each other in order to solve common problems (Bellifemine et al., 2007); (Rudowsky, 2004). Agent oriented paradigm is the appropriate practice for modelling, design and implementation of complex and distributed software systems (Jennings, 2001). Agents execute inside execution environment or agent framework that controls agent's life cycle and provides necessary mechanism for agent's execution. Agent is capable of flexible, autonomous action in that environment in order to meet its designer objectives (Bellifemine et al., 2007); (Wooldridge, 1997).

When single agent is incapable of solving a problem, due to its limited capability or knowledge, agents join together forming distributed loosely coupled network to solve their problems. Such community is called Multi – agent system (Sycara, 1998). In MAS task is decomposed, sub-tasks are divided among agents, and agents interact with each other. MAS is applicable in a wide range of problems, such as information retrieval (Punithavathi and Duraiswamy, 2010), mobile telecommunication networks (Jurasovic et al., 2009), power supply management (Yang et al., 2006) (Zhang et al., 2004), space exploration and other (Rudowsky, 2004).

Agents and their resources may become unavailable due to the machine crashes, communication breakdowns, process failures, and numerous other hardware and software failures (Fedoruk and Deters, 2002). Significant part of the research effort regarding MAS failure handling is oriented towards increasing the robustness and failure resistance of agents themselves (Mitrović et al., 2010). In today's connected world where quality of service is of extreme importance and level of service is formally defined through Service Level Agreements (SLA) (Anon, 2002), a system is called a high – assurance system, when heterogeneous and

Stanković R. and Štula M..
Fault Tolerance through Interaction and Mutual Cooperation in Hierarchical Multi-Agent Systems.
DOI: 10.5220/0004182003370344
In *Proceedings of the 5th International Conference on Agents and Artificial Intelligence* (ICAART-2013), pages 337-344
ISBN: 978-989-8565-38-9

changing requirement levels of the Quality of Service (QoS) are satisfied (Ahmad et al., 2001) (Ahmad et al., 2003). High assurance of service in distributed multi – agent systems is a key feature for its successful deployment.

Despite considerable efforts spent on developing multi – agent systems, actual number of deployed systems is not nearly as large; one of the main reasons for that are failure handling issues, which make MASs very brittle (Briot and Ghédira, 2003). Techniques used in traditional distributed architectures are often static, require special infrastructural support and restrict functionality of agent frameworks (Kumar and Cohen, 2000). When a fault occurs in MAS, interactions between the agents may cause the fault to spread throughout the system in unpredictable ways (Almeida, et al., 2006). This is extremely undesirable in critical applications, where the occurrence of a fault may cause the loss of lives, delays in the products manufacturing process (business loses) or suboptimal resource utilization.

In this paper, we introduce our solution to building reliable MAS. It is based on the agent interaction that enables fault tolerance in the hierarchical MAS regardless of an agent framework used; unlike other solutions that are framework dependent (Almeida et al., 2006), (Faci et al., 2006), (Helsinger et al., 2004). We introduce special class of agents, called Arbiters, that help ordinary working agents recover in case of faults. Our approach relies on agent communication and cooperation to enable successful fault recovery.

The remainder of this paper is organized as follows. In Section 2 we provide an overview of the related work. Section 3 describes our approach to building the fault tolerant multi – agent system. Section 4 shows comparative simulation results. Finally, Section 5 summarizes our approach and on-going work.

2 RELATED WORK

Research on the agent based failure handling has been extensive; it can be classified to a failure handling by individual agents within an agent framework and a failure handling by an agent framework (*Figure 1*). In this overview we will show different researchers approaches used to tackle failures in agent based systems, their key features and weaknesses.

Failure handling by individual agents within the agent framework is reported in the following works. Kumar et al. propose fault tolerance (FT) architecture based on agent brokers. (Kumar et al., 2000). Approach arranges brokers in hierarchical teams, which are then used for communication and coordination among agents. Fault tolerance mostly concentrates on the fault tolerance of brokerage team and not on individual agents doing actual work. Proposed architecture requires extra computing for the management of brokerage layers.

Proxy based approach is proposed by Fedoruk et al., in which a proxy transparently handles the replication group based on predefined policies (Fedoruk and Deters, 2002). The proxy is nothing more than a computational entity, which provides interface to a set of agent replicas. This approach suffers from the centralized bottle neck by proxy itself and only concentrates on replicating agents of a multi - agent system. It is more costly in terms of forming replication groups of all the agents in a multi - agent system. The approach also lacks reusability in particular concerning the replication control.

Xu et al. address fault detection techniques proposed in (Xu and Deteis, 2005), (Dellarocas and Klein, 1999) by sending periodic events to agents inspecting their state. Separate query language to inspect agent state and action language to recover agent in error are used. This kind of exception handling requires periodic probing and is quite an overhead on the system. Agent recovery is vaguely described.

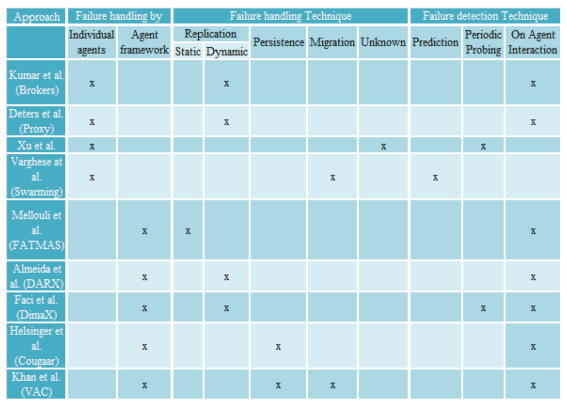
Varghese at al. propose an agent based approach that handles single node failures for parallel summation algorithms in computer clusters. (Varghese et al., 2010) (McKee and Varghese, 2010). The agents intercommunicate across computing nodes, when they detect failure they move away from the faulty node notifying other agents to stay away from the problematic node. Solution is narrowly specialized and highly dependent on accurate prediction; in case of unpredicted failure calculations must be done again.

Failure handling by the agent framework is focused on maintaining highly available MAS that can withstand numerous agent, hardware and network failures is reported in the following works. First we will look at researches focused on ensuring failure handling using agent replication. The simplified idea behind these techniques is to keep multiple copies of an agent, distributed across a number of nodes. In case the original agent fails, one of the copies automatically takes over its task execution process. Approaches evolve on how to define critical agents that should be replicated to minimize usage of system resources. FATMAS methodology introducing guidelines for the analysis and the design of fault – tolerant MAS is proposed by Mellouli et al. (Mellouli et al., 2004). Moreover, it provides guidelines for an agent and task replication, which enables reduction in the replication cost. However, the approach addresses to closed MAS; agent criticality is defined at design time, meaning replication is static. Almeida et al. (Almeida et al., 2006), (Marin, 2003) report the implementation of the Dynamic Agent Replication Extension (DARX), a failure handling agent framework. In this model, failure handling is performed by replicating those agents that are critical to the system and whose future plans could influence other agents in the system. Agent's criticality is dynamically evaluated by revising agent's plans and roles. Faci et al. introduced agent framework based on DARX named DimaX (Faci et al., 2006). It uses interdependence graphs for evaluating the significance of every agent in the system, and uses DARX to accordingly maintain an optimum number of replicas (process known as adaptive replication). The system also employs host monitors, low-level agents that try to perform early detection of failures and inform all interested parties about the problem in a timely manner. This would, for example, allow an agent and its replicas to leave the troubling environment, or to move important data from it. At the basis of the monitoring is a heartbeat technique, used by the agents in the system to indicate their valid operational status to each other. When determining agent criticality, no guarantee is given that the agent is truly critical, it can only seem critical to the system; monitoring technique provides extra communication overhead.

In the MAS fault handling research focus is not only the agents. Helsinger et al. propose Java-based architecture for the construction of large-scale distributed agent based applications named Cougaar (Helsinger et al., 2004). Cougaar is designed to be continuously available and degrades gracefully when its components get disconnected or damaged. Agent is viewed as a set of problem solving behaviours interacting via traditional blackboards with standard publish/subscribe semantics. If an agent is unable to contact a member of its community it can send a health alert message to a health monitor agent, which is responsible for the recovery of agents. Recovery can be done in two ways; either by retrieving an appropriate community state needed to pursue the problem solving or by rejoining the failed agent to its community which has begun a new problem solving stage. Approach is adaptable, however lacks compliance to the standard and is extremely complex; no guarantee is given that the MAS will correctly pursue its goals after agent failures e.g. failures can cause deadlocks and other problems.

Khan et al. propose approach based on decentralized architecture (Khan et al., 2005). Virtual agent cluster (VAC) is an abstraction which provides scalable transparency among all the distributed machines over which the agent platform is deployed. It enables seamless agent migration and communication between different machines in the virtual cluster. This approach introduces load balancing and agent platform fault tolerance in distributed MAS, meaning even if one machine in platform is operational, MAS is stable and the agents can execute. Agent states are preserved using persistence which is resource costly; after agents are restored, they rejoin community which can have different state then when the agent failed. This issue is not resolved.

Failure handling by agent framework offers fault handling features of certain quality and is applicable for some fault scenarios, but at the same time it neglects some other features of agent framework and introduces large degree of complexity. As a consequence interoperability between heterogeneous multi – agent systems is difficult if not impossible to realize (D. Mitrović et al., 2011); (Suguri et al., 2002).



Approach	Failure handling by		Failure handling Technique					Failure detection Technique		
	Individual agents	Agent framework	Replication		Persistence	Migration	Unknown	Prediction	Periodic Probing	On Agent Interaction
			Static	Dynamic						
Kumar et al. (Brokers)	x			x						x
Deters et al. (Proxy)	x			x						x
Xu et al.	x						x		x	
Varghese at al. (Swarming)	x					x		x		
Mellouli et al. (FATMAS)		x	x							x
Almeida et al. (DARX)		x		x						x
Faci et al. (DimaX)		x		x					x	x
Helsinger et al. (Cougaar)		x			x					x
Khan et al. (VAC)		x			x	x				x

Figure 1: Failure handling classification.

3 COOPERATIVE FAULT TOLERANCE

We propose cooperative fault tolerance (CFT) solution that can easily be included into any existing agent framework. Unlike other approaches, we do not base our solution on replication (Mellouli et al., 2004), (Deters et al., 2002), (Almeida et al., 2006) or complex framework implementation (Faci et al., 2006), (Helsinger et al., 2004); instead our approach uses agent cooperation to enable fault tolerance in the hierarchical MAS. Agents in the MAS can organize differently to solve problems (Mladenović, 2011). Our solution focuses on the hierarchical MAS. Hierarchy is crucial part of many complex systems (Ravasz et al., 2003). It is an arrangement of items (objects, names, values, categories, etc.) in which the items are represented as being "above," "below," or "at the same level" as other items. This type of organization prevails in big complex systems like companies, governments, cosmology, religions and technical systems (Ravasz et al., 2003) (Torrel et al., 2007). Organization of MAS can be displayed with a graph; graph consists of two types of information: vertices (nodes) and edges (connections that display node interaction) (Wilson, 1990). Hierarchy can be represented with a tree graph; a tree is an undirected graph in which any two vertices are connected by exactly one simple path. In other words, any connected graph without cycles is a tree (Wilson, 1990).

Our solution divides agents into two categories: worker agents (Workers) and arbiter agents (Arbiters). Workers represent typical agents in a MAS that solves different user tasks (Bellifemine et al., 2007) (Rudowsky, 2004). Worker does everything in its power to solve the given task; if it is incapable of solving the problem due to its limited capability or knowledge, it organizes into hierarchies and communicates with other agents to solve it. Arbiters are committed to solving failures that occur when Worker agent/s fail. To maximize resource utilization initially only few instances of Arbiters exist. Main instance is called "Distributor" which receives initial error notification from Worker agents and forwards those to the specific Arbiter dedicated to resolve that specific agent failure. Each arbiter is dedicated to solving a single problem. Distributor is also in charge of maintaining optimal number of Arbiters. If there are more failures than free Arbiter agents, it creates additional agents. Once the failures are resolved Distributor disposes of additional arbiter agents, maintaining original number of arbiters in a failure free system. In case of the Distributor failure, first Arbiter that registers its failure notifies other Arbiters via multicast message and takes it place. New Distributor then builds up the list of available Arbiters and their tasks, completing the transition process. Only agent that is not protected by our approach is the root agent which can be protected by the basic replication approach. For example, let us define hierarchical MAS as the graph $G = \{r, w, x, y, z\}$ where vertices are related in such a way that the *r* is the root node; *w, x* are its child nodes and *y, z* nodes are the children of *x* (*Figure 2*).

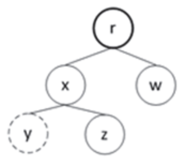


Figure 2: MAS hierarchy structure.

Let us define set of attributes that accurately describe node tasks with the list of ordered triple: *(<'name1', 'task1', 'solution1'>, <'name2', 'task2', 'solution2'>...)*. For agent represented by node *x* original attribute (task) received from its parent is defined as $atrX_{orig}$. Original task is defined as set $T=\{atrX_1,atrX_2,atrX_3\}$; $atrX_1$ and represents task that agent *x* has to do itself. $atrX_2$ and $atrX_3$ represent tasks forwarded to agents *y* and *z* respectively. Algorithm that splits task into subtasks can be defined as function $f{:}T\rightarrow S$ that is invertible, meaning there is a function g with domain S and range T, with the property: $f(t)=s$ if and only if $g(s)=t$; this is important so that agent can check consistency when recreated. Tasks and sub tasks have hierarchical organization.

Relation R, which binds set of sub tasks into the original task (Johnson, 2009) exists. To prove that R binds set of sub task into the complex task we define relation as: $R{:}\left\{atrX_1,atrX_2,atrX_3\right\}\rightarrow$ $<atrX_1,atrX_2,atrX_3;atrX_{orig};R>$. This relation can be displayed via hierarchical cone, where the base of the cone is formed by sub tasks while the tip of the cone represents main task $atrX_{orig}$. If sub task level is defined $Level\ (atrX_i)$, then we can write $Level\ (atrX_I) < Level\left(atrX_{orig}\right)\ \forall\, atrX_I$, meaning that the task cannot be part of the sub task. This principle defines absolute measure of difference between levels in complex systems. Basic operations on the nodes of the graph (insert, delete) are defined as:

```
Insert(y, tasklist
(<'name1','task1','solution1'>,
<'name2','task2','solution2'>...),
x)
```

This command inserts node *y*. If the node does not exist, it creates the node with appropriate attributes. First attribute in the list represents original task. If the node exists, new attributes overwrite old ones and connection is made to the parent node *x*.

```
Delete(y,  tasklist, x)
```

This command deletes mentioned attributes and relation between node *y* and its parent *x*. If no attributes are mentioned, only relationship between nodes is removed. If no attributes are mentioned and parent node is mentioned, then node *y* is deleted with all its attributes and relation to its parent.

In the case of error Arbiters are used to repair agents in error. Let us suppose that the agent *x* fails and its child agent y detects that while returning a result of its task. Arbiter *Ar* is assigned to solve the problem, it collects information about agents aware of agent *x* and their attributes. Now Arbiter *Ar* has all data necessary to recreate agent *x*.

We can define attributes that Arbiter *Ar* collected as $atrAr=\{atrX_{orig},atrX_2,atrX_3\}$. Worker *x* is recreated with agents sub tasks mapped to the appropriate existing child agents. Sub task that is missing is $atrX_1$ and agent assigns it to itself. Because of this, recreated node *x* will reuse result of child Workers *y* and *z*; only sub task $atrX_1$ will be redone, if it was completed, if it was not completed, it will be only executed once so no impact on performance will occur. Operations that Arbiter can do (find, recreate, notify) are defined as:

```
Find(x)
```

This is a multicast message to which all agents in relation to Worker *x* will answer. Arbiter *a* will collect the list of Workers *x* was related to, its original tasks and sub tasks and forwarded it to its children.

```
Recreate(x,tasklist
(<'name1','task1','solution1'>,
<'name2','task2','solution2'>...),
agentlist(r,y,z))
```

This command recreates node *x* as a child node of *r* and as a parent node of nodes *y* and *z* with set of attributes that accurately describe the node tasks. First attribute in the attribute list represents original task, first node in the node list represents parent node.

```
Notify(x)
```

This is multicast message that notifies all agents in relation to *x* that the Worker *x* is recreated and running. Timing of the error can be a crucial factor that diminishes benefits of this approach. Let us get back to *Figure 2* and suppose that Workers *y* and *z* finished their task and returned result to their parent node; *x* finishes its calculation and fails at exact that time. If we do not do anything, Workers *x, y* and *z* would be recreated and all tasks redone. To tackle this problem we propose result parsing mechanism. After task is completed result is passed up in the hierarchy. Each agent in the hierarchy has a memory buffer in which it stores results received from its children, grandchildren and so on. Result parsing has its benefits in following scenarios: providing extra tolerance to multiple failures in the same branch of the execution tree and quick recovery of the parent whose children completed work without extra calculation. When agent *y* and *z* finish and report their results, data is stored in their parent *x* but also grandparent r and if needed further up the hierarchy. With result parsing if Worker *x* fails after its children reported the results there is no problem, since *y* and *z* results are stored in Worker *r*. Worker *r* notices that *x* is in error and notifies Arbiter agent. Worker *x* is restored as before but in the restore command are also results from workers *y* and *z*. Worker *x* then combines result from Workers *y* and *z*, does its task and returns result to its parent *r*. Parsing operation is defined as:

```
Parse(x,tasklist
(<'name1','task1','solution1'>),
, y)
```

Worker *x* uses this command to parse data from node *y* forward up the hierarchy.

4 TESTING AND EVALUATION

In this section we evaluate viability of CFT solution in comparison to the basic no fault tolerant scenario and to the active replication DARX like framework. Simulations where created using NetLogo, a multi – agent programmable modelling environment, used for simulating natural and social phenomena. NetLogo is particularly well suited for modelling complex systems developing over time. Modellers can give instructions to hundreds or thousands of "agents" all operating independently (Wilensky, 1999). This makes it possible to explore the connection between the micro – level behaviour of individuals and the macro – level patterns that emerge from their interaction.

Simulations were comprised in such a way that they test solutions through the whole working envelope by varying task complexity and error probability; the more complex the task the longer simulation will run (e.g. length 80 means that simulation will divide task into sub tasks for 80 consecutive steps, task complexity is greater and total number of agent increases. Simulation lasts for much longer than length of division, until final result

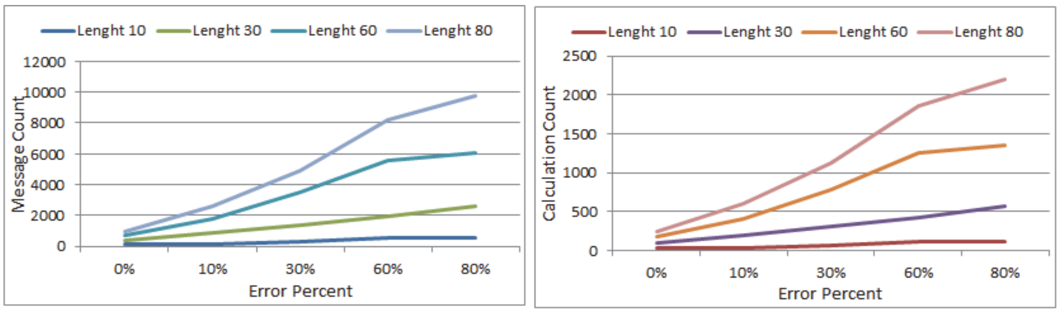


Figure 3: Basic no FT solution, data for various simulation length and error probability.

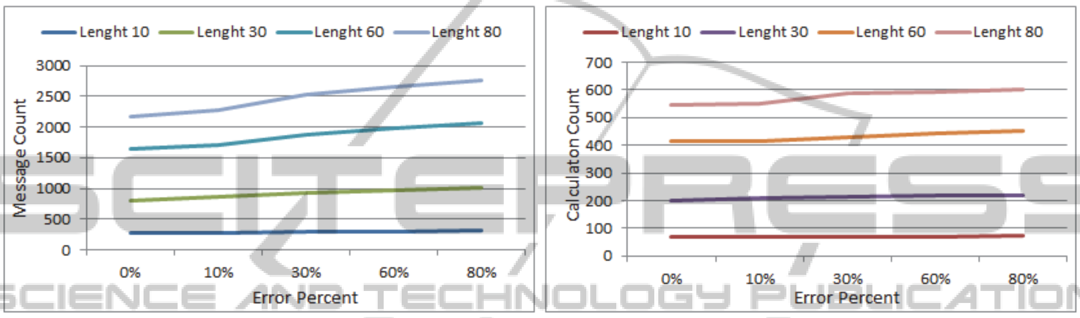


Figure 4: DARX like replication solution, data for various simulation length and error probability.

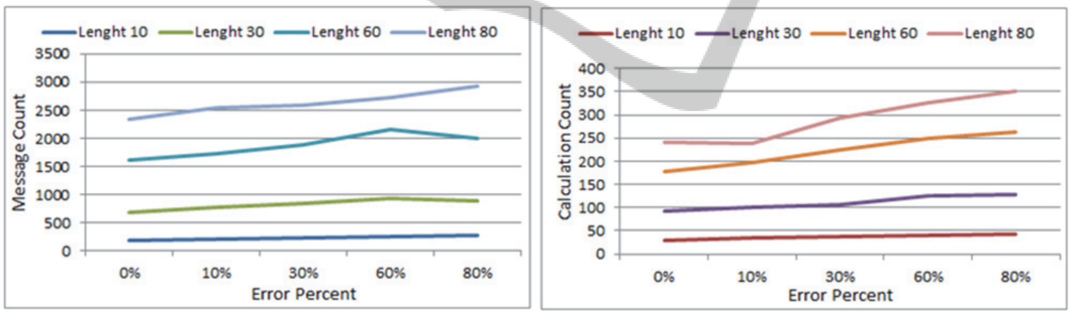


Figure 5: CFT solution, data for various simulation length and error probability.

is returned to the user). For every solution baseline information was obtained by running the simulation with no errors for various task sizes.

For each algorithm/complexity pair ten simulation runs where performed; average values for each run are presented. Numerous parameters where monitored, but the focus of our tests were on the total message count (network load) and total number of calculations done (CPU load). Those parameters where selected as they represent most significantly load on the system.

Total number of messages exchanged in the system is sum of real messages needed for normal task execution and helper ones needed for failed task recreation, fault tolerance and replication if it is used. Total number of calculations done by the system is sum of normal calculations done by agents, extra calculation done when node fails and extra calculation done by replicated nodes if replication is used.

For the no FT solution (Figure 3) situation is quite straightforward; number of messages and calculations grow as the number of agents needed to solve the task grows; as the error probability increases, so does the number of messages and calculations. This is the most efficient scenario when there are no errors in the system, but even the slightest error probability sky rockets number of extra messages and calculation, stressing importance of the fault tolerant solution.

Results for replication solution are shown in Figure 4; we simulated DARX like active replication mode using available sources (Almeida et al., 2006), (Marin, 2003) for reference to make the simulation results more accurate. It is important to notice that the message and calculation count has only the slight increase regardless of error probability, remaining pretty much the same throughout the increase of task complexity. This solution excels over the basic no FT solution in all except "no error" scenario.

Proposed CFT solution results are shown on Figure 5. Our solution excels over the no FT solution. In no error scenario our solution has a bigger message count than no FT scenario because of extra parsing messages, while number of calculations is the same meaning our approach does not waste processor time; As the error probability increases our solution achieves on average five times less number of calculations, for all group sizes than no FT scenario. Our message count results are comparable to the DARX like active replication solution (slightly better for all but the largest task complexity). Comparing number of calculations our solution excels over results achieved by active replication solution, number of calculations are two times lower for all group sizes and error probabilities.

5 CONCLUSIONS

Multi-Agent Systems are well suited for development of complex, distributed systems; therefore usage of the MAS is expected to grow in the future. Fault tolerance is an important property of large scale MAS which enables successful application of the system. In recent years a lot of research has been done addressing specific aspects of fault tolerance, however none of the proposed solutions gives a complete coverage of the problem leaving a lot of opened areas for new approaches and improvements. After extensive overview of the available FT solutions in section 3 we proposed a novel solution for creating fault tolerant MAS. Proposed solution can be used as a set of implementation guidelines that enable its application to any agent framework, providing that agents form hierarchies when solving problems. Our approach enables: easy substitution of agent in error by replacement agent, preservation of child agent's data so no extra work is necessary. This enables MAS to recover quickly and without wasting valuable system resources, ensuring that agents continue to pursue their goals fast and correctly.

In section 4 we provided extensive simulation and compared results between our solution, basic no FT scenario and active replication solution. We focused on the two main parameters that represent most significantly load on the system, calculation count (CPU load) and message count (network load). Simulation results show advantages of our solution and proves that our solution enables performance light FT for hierarchical MAS regardless of agent framework. We should point out the fact that our solution is easy to implement in any existing or future MAS regardless of framework in use while most of other FT solutions have high degree of complexity and are often related to specific agent framework. This means that complexity and ease of use should be important factor in any future comparisons.

Further research is directed to free our solution from its limited usage to MAS that form hierarchies while solving problems. This would enable the solution to be used in any type of MAS. Basic idea behind this direction is to use peer-to-peer (P2P) communication between unrelated agents in the MAS, so agents can save their state among their peers. In the case of error Arbiters can recreate agent in error with the help of data stored at its peers. To successfully improve current solution we must overcome number of challenges. Most prominent of them is the optimal usage of P2P communication so the system would not suffer performance degradation due to the intensive messaging. Second challenge is to develop viable incentive model that will encourage agents to give up part of their resources for unrelated peers pursuing unrelated goals. Third challenge is to ensure that the solution has distinct advantage over other FT solutions in parameters that we see as relevant in the MAS now (CPU load, network load, overall complexity and ease of use) and others that we may include in the future.

REFERENCES

Ahmad, H. F., Sun, G. and Mori, K., 2001. *Autonomous Information Provision to Achieve Reliability for Users and Providers.* s.l., IEEE Proc. of the Fifth International Symposium on ADS (ISADS01), pp.65-72.

Ahmad, H. F., Sun, G. and Mori, K., 2003. *Dynamic Information Allocation Through Mobile Agents to Achieve Load Balancing in Evolving Environment.* s.l., IEEE Proc. of the Sixth International Symposium on ADS (ISADS03), pp.25-33.

Almeida, A. d. L., Aknine, S., Briot, J.-. P. and Malenfant, J., 2006. *Plan-based replication for fault-tolerant multi-agent systems.* s.l., Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium.

Anon., 2002. *SLA Information Zone.* [Online] Available at: http://www.sla-zone.co.uk/ [Accessed June 2011].

Bellifemine, F. L., Caire, G. and Greenwood, D., 2007. *Developing multi-agent systems with JADE.* s.l., John Wiley & Sons, Inc..

Briot, J.-P. and Ghédira, K., 2003. *Déploiement des systemes multi-agents.* s.l., Revue des Sciences et T echnologies de l'Information, hors série/JFSMA 2003.

D.Mitrović, Ivanović, M., Budimac, Z. and Vidaković, M., 2011. *An overview of agent mobility in heterogeneous environments.* s.l., Proceedings Of The Workshop On Applications Of Software AgentS.

Dellarocas, M. and Klein, C., 1999. *Exception Handling in Agent Systems.* s.l., Proceedings of the Third International Conference on Autonomous Agents, Seattle, WA.

Faci, N., Guessoum, Z. and Marin, O., 2006. *DimaX: a fault-tolerant multi-agent platform.* s.l., In Proceedings of the 2006 international workshop on Software engineering for large-scale multi-agent systems, pp. 13- 20.

Fedoruk, A. and Deters, R., 2002. *Improving fault-tolerance by replicating agents.* s.l., In Proc. AAMAS-02, pp. 737-744, Bologna.

Helsinger, A., Thome, M. and Wright, T., 2004. *Cougaar: a scalable, distributed multi-agent architecture.* s.l., In Preccedings of International Conference on Systems, Man and Cybernetics pp. 1910–1917.

Jennings, N. R., 2001. *An agent based approach for building complex software systems.* s.l., Communication of the ACM, 44(4) pp. 35-41.

Johnson, J., 2009. Hypernetworks of Complex Systems; Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. *Complex Sciences,* 4(pp. 364-375).

Jurasovic, K., Kusek, M. and Jezic, G., 2009. *Multi-agent service deployment in telecommunication networks.* s.l., In Agent and multi-agent systems: technologies and applications, LCNS Springer Berlin / Heidelberg, pp. 560 — 569,.

Khan, Z. et al., 2005. *Decentralized architecture for fault tolerant multi agent system.* s.l., In Proc. of Autonomous Decentralized Systems (ISADS), pp. 167-174..

Kumar, S. and Cohen, P., 2000. *Towards fault-tolerant multi-agent system architecture.* s.l., In proceedings of the fourth international conference on Autonomous agents, pp459 – 466.

Kumar, S., Cohen, P. R. and Levesque, H. J., 2000. *The adaptive agent architecture: Achieving fault-tolerance using persistent broker teams.* s.l., In Proceedings of the 4th International Conference on Multi-Agent Systems, Boston, MA.

Marin, O., 2003. *The DARX Framework: Adapting Fault Tolerance For Agent Systems.* s.l.:THÈSE DE DOCTORAT DE L'UNIVERSITÉ DU HAVRE.

McKee, B. and Varghese, G., 2010. *Exploring Carrier Agents in Swarm-Array Computing.* s.l., Scalable Computing: Practice and Experience, Volume 11, pp. 53–62.

Mellouli, S., Moulin, B. and Mineau, G., 2004. *Towards a modelling methodology for fault-tolerant multi-agent systems.* s.l., In Informatica Journal 28, pp. 31–40.

Mitrović, D., Budimac, Z. and Vidaković, M., 2010. *Improving Fault-Tolerance of Distributed Multi-Agent Systems with Mobile Network-Management Agents.* s.l., Proceedings of the International Multiconference on Computer Science and Information Technology pp. 217–222.

Mladenović, S., 2011. *Interoperability in hierarchical and heterogeneous systems.* s.l.:Doctoral thesis, Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture in Split.

Punithavathi, R. and Duraiswamy, K., 2010. *A fault tolerant mobile agent information retrieval system.* s.l., In Journal of computer science, Vol. 6, pp. 553 - 556.

Ravasz, E. and Barabási, A. L., 2003. Hierarchical organization in complex networks. 67(2).

Rudowsky, I., 2004. *Intelligent Agents.* New York, Proceedings of the Americas Conference on Information Systems, New York.

Suguri, H., Kodama, E., Miyazaki, M. and Kaji, I., 2002. *Assuring Interoperability between Heterogeneous Multi-Agent Systems with a Gateway Agent.* s.l., Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering.

Sycara, K. P., 1998. *Multiagent Systems.* s.l., AI Magazine, American Association for Artificial Intelligence.

Tanenbaum, A. S. and Steen, M. v., 2002. *Distributed Systems: Principles and Paradigms.* Upper Saddle River, New Jersey 07458: Prentice Hall.

Torrel, J.-.C., Lattaud, C. and Heudin, J. -. C., 2007. *Complex Stellar Dynamics using a hierarchical multi-agent mode.* Erice, Italy, Modelling and simulation in science, Proceedings of the 6th International Workshop on Data Analysis in Astronomy pp.307-312.

Varghese, B., McKee, G. and Alexandrov, V., 2010. *Handling single node failures using agents in computer clusters.* s.l., International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS).

Wilensky, U., 1999. *NetLogo Home Page.* [Online] Available at: http://ccl.northwestern.edu/netlogo/ [Accessed 06 May 2012].

Wilson, R. J., 1990. *Graphs and heir use.* s.l.:New Mathematical Library.

Wooldridge, M., 1997. *Agent based Software Engineering.* s.l., IEE Proceedings of Software Engineering 144, pp: 26-37.

Xu, P. and Deteis, R., 2005. *Fault Management in Multi Agent Systems.* s.l., In Proceedings of Symposium on Applications and the Internet.

Yang, Z. et al., 2006. *A multi-agent framework for power system automation.* s.l., In International journal of innovations in energy systems and power, Vol. 1, No. 1.

Zhang, Z., McCalley, J. D., Vishwanathan, V. and Honavar, V., 2004. *Multiagent system solutions for distributed computing, communications, and data integration needs in the power industry.* s.l., In Proceedings of the General Meeting of the IEEE Power Engineering Society, IEEE Press, pp. 45 - 49.