Proceedings of the International Multiconference on Computer Science and Information Technology pp. 173–180
ISBN 978-83-60810-27-9
ISSN 1896-7094

BeesyBees – Efficient and Reliable Execution of Service-based Workflow Applications for BeesyCluster using Distributed Agents

Paweł Czarnul, Mariusz Matuszek, Michał Wójcik and Karol Zalewski
Faculty of Electronics Telecommunications and Informatics, Gdansk University of Technology
Email:{pczarnul,mrm}@eti.pg.gda.pl

***Abstract*—The paper presents an architecture and implementation that allows distributed execution of workflow applications in BeesyCluster using agents. BeesyCluster is a middleware that allows users to access distributed resources as well as publish applications as services, define service costs, grant access to other users and consume services published by others. Workflows created in the BeesyCluster middleware are exported to BPEL and executed by agents in a distributed environment. As a proof of concept, we have implemented a real workflow for parallel processing of digital images and tested it in a real cluster-based environment. Firstly, we demonstrate that engaging several agents for distributed execution is more efficient than a centralized approach. We also show increasing negotiation time in case of too many agents. Secondly, we demonstrate that execution in the proposed environment is reliable even in case of failures. If a service fails, a task agent picks a new equivalent service at runtime. If one of task agents fails, another of remaining agents takes over its responsibilities. The communication between the middleware, agents and services is encrypted.**

I. Introduction

INTEGRATION of various systems and components is one of the most crucial challenges that needs to be addressed today. In service-based environments this requires efficient algorithms for scheduling workflows i.e. selection of services for particular tasks while meeting optimisation goals. In the literature [1], [2] the workflow is usually defined as a directed acyclic graph $G(V, E)$ where vertices V denote tasks to be executed while edges denote time dependencies between the tasks. A workflow can be concrete if there is one service to be executed for each task or abstract, in which case there is a set of functionally equivalent services for each task, possibly with different QoS parameters. The latter almost always include execution time and cost and if needed others, such as reliability, reputation of the provider etc. One service needs to be chosen for each task such that a global goal is optimized with possibly meeting other QoS constraints. Typical optimisation goals include minimisation of execution time with an upper bound on the total cost of selected services which makes the problem NP-hard. Such workflows are executed by workflow engines which invoke services, control their statuses and transfer data.

II. Problem Statement

Workflows represent an integration of tasks, in which a set of alternative services may be defined (either manually or automatically) for each task. Alternatives may be statically linked to a task or dynamically assigned at runtime. Regardless of which service is selected for a task, the workflow should be executed in such a way that efficiency and reliability of workflow execution should be maximised. We assume that the workflow definition as well as the optimisation criteria and solver are already given [3] which means that services selected for particular tasks are known.

At this point, the problem becomes how to manage the execution of a distributed workflow application so that the following goals are optimized and maintained:

- efficiency — because of potentially high communication latency and low bandwidth of WANs, communication between services through agents close to interacting services is faster than through a distant centralised execution engine;
- reliability — since the node(s) on which the workflow execution engine runs as well as service locations may be geographically distributed, it is likely that at times connections may be lost; the solution should ensure reliable and continuous execution in such cases;
- security — all service invocations and data flows should be performed in a fully secure manner.

We present an agent-based distributed management mechanism that addresses all of the above and examine its performance in a practical example executed in a real environment.

III. Related Work

A. Workflow Types and Workflow Management Environments

Literature studies bring examples of many middlewares and environments for workflow execution. First and foremost, these do differ based on the target workflows to be executed. Scientific workflows usually focus on integration of computational services into chains executed on HPC resources such as clusters and supercomputers, or analysis of data acquired from input sensors or devices by services installed on powerful machines. Such workflows can span over various geographically distributed sites and virtual organisations forming grid systems [4], [5] and usually focus on data flow. There are several systems [4] that support such applications including Kepler [6], Gridbus, Triana [7], Pegasus [8], P-GRADE [9], Directed Acyclic Graph Manager (DAGMan), ICENI [10], UNICORE, Taverna, GridFlow, GrADS, Askalon, GridAnt.

Conversely, business workflows focus on discovery, selection and integration of services offered by various parties so that certain QoS parameters are met. Such workflows focus more on controlling the flow and often incorporate constructs such as choice, loop and other conditionals. Business workflows usually consider many more QoS parameters apart from the execution time and cost such as reliability, accessibility, fidelity, conformance, security etc. Systems such as Meteor-S [11], [12] focus on automating the process of service discovery and selection using an ontology-based approach. Selected services are executed automatically making service discovery, selection and execution almost fully automatic, based on the given workflow specification and given services and their descriptions. BPEL [13] is often used for describing business oriented workflows and includes control and data flows as well as service invocations. There are several execution engines for workflows specified in BPEL such as The ActiveBPEL Engine [14], bexee [15] or Silver [16].

Thirdly, in the context of pervasive and ubiquitous computing, workflows often react to events asynchronously and more importantly the context is considered for invocation of a service and changing the state. The latter is defined as information defining the state of an object [17]. As an example, uWDL [18] is used for describing ubiquitous workflows and allows specification of both the services as well as the context and service flow through the `node` and the `link` elements respectively.

B. Existing Service-based Solution in BeesyCluster

The already existing workflow support module in BeesyCluster developed by our research group [3], [19] contains a workflow editor, an optimiser and execution engine and has already been tested on a variety of scientific and business workflow applications, such as multimedia processing, numerical simulations [19] or business workflows [3].

BeesyCluster is a middleware that allows its users to invoke sequential or parallel applications exposed as services on registered system accounts on various clusters and servers. Such services can be assigned to particular workflow tasks in BeesyCluster's editor. The workflow editor is implemented by an applet. Created workflows are saved in BeesyCluster's database. Based on service execution times and costs, BeesyCluster's optimiser selects one service for each task so that the given criteria are optimised. It can optimise either a linear combination of the total cost of selected services and the workflow execution time or e.g. the execution time with a global constraint on the total cost of selected services. The execution engine is implemented within a Java EE server. For each workflow node a `SIMessageBean` which is a message driven bean is responsible for executing the service chosen for a particular workflow task by the optimizer is used. This is done within the `onMessage` method which is executed upon receiving a JMS message. For the given task, after the service has been executed, the bean copies output files to dedicated directories of services chosen for successor tasks and initiates execution of following tasks by sending JMS messages. The execution status of a particular node instance is updated in the database dynamically.

The drawback of this solution is centralized management of execution. If large data is passed between services, it needs to be passed through BeesyCluster which increases the workflow execution time. We propose to optimize this by launching several distributed JADE agents that would launch services for workflow tasks and act as local proxies for communication between the services. Comparison to other agent-based approaches is presented in Section V.

IV. Proposed Agent-based Solution

As stated in the previous section, BeesyCluster's execution engine is implemented within a Java EE server. This approach allows for easy execution management, but it can also create a bottleneck for efficient execution and a single point of failure. Should a problem with either the server or its network connectivity occur, the whole workflow execution may be at risk. This in itself is a serious limitation.

By using a set of well defined, industry standard conformant interfaces, we migrated the task of execution of a prepared workflow to an agent-based environment, which we nicknamed *BeesyBees*. By separating the two tasks we also gained a possibility of having a pluggable architecture, allowing for experimentation with different approaches to workflow execution. By implementing the execution management in mobile agents we take advantage of enormous flexibility of this environment.

In this paper, we concentrate on increased efficiency and reliability of the workflow execution, which is a main difference to original BeesyCluster's solution.

A. Architecture

The BeesyBees system is based on the JADE (Java Agent DEvelopment Framework) [20] agent system implementing the FIPA [21] communication and the OMG MASIF [22] management standards. It provides distributed and decentralized workflow execution for BeesyCluster using agents.

BeesyCluster itself can be regarded as a middleware that offers an easy-to-use WWW interface to access various accounts on various geographically distributed clusters and servers through a single account in BeesyCluster. This allows management of files and directories on such clusters and servers, compiling and running sequential and parallel applications with support for various queuing systems, a team work environment with sharing data and others. Furthermore, users can publish own applications as services within BeesyCluster and make them available to other BeesyCluster users. A provider can specify a cost the client would need to pay for invoking the service. BeesyCluster contains a subsystem for handling virtual payments between users. Access to clusters and servers as well as running particular commands, searching for and running services is also possible through the Web Service interface [23], [24].

Figure 1 shows a generalised diagram for BeesyBees system architecture. There are four types of agents implemented in the system:

- GateWayAgent — receives workflow descriptions in BPEL and spawns TaskAgents,
- GraphAgent — monitors the progress of workflow execution and gives a graphical representation of agent instances and decisions made,
- StatusAgent — collects reports on task execution states from TaskAgents and persist actual status of workflow realization,
- TaskAgent — executes a single workflow task. A group of TaskAgents executes the workflow cooperatively.

A workflow is composed using BeesyCluster's SIEditor applet, a tool for modeling workflows that consist of tasks with services assigned out of those defined in BeesyCluster. For the each task from one to few services can be chosen. First, the definition of the workflow to be run is fetched from BeesyCluster's database and saved in BPEL, which then is handed to the BessyBees system where it is received by the GateWayAgent. All the communication between BeesyCluster and BeesyBees is handled with appropriate web services. Then the GateWayAgent spawns TaskAgents in order to execute workflow's tasks.

TasksAgents negotiate the assignment of each task [25]. TaskAgent interested in executing a particular task sends its proposal including its matching score to all other agents. The score is an object which can be filled with various comparable metrics, like for example a load of a machine an agent resides on. When the agent receives other agent's execution proposals it agrees only when it is not interested in that particular task or it's matching score is lower. This process is represented in Figure 2. Finally, the TaskAgent which received approvals only, begins task execution. When the task is done the TaskAgent sends notification to all agents executing particular workflow, containing information which task has been done and using which service. Example of notification is presented in Figure 3.

The execution state of workflow's tasks is monitored by a StatusAgent. Messages sent by TaskAgents during workflow execution are monitored. These include information whether a task has been executed, or if there were problems with calling services. That data is saved persistently by the StatusAgent so it could possibly be used for recovery after the whole system crash. Additionally a GraphAgent which shows a graphical representation of a workflow execution can be turned on.

B. *Efficiency*

The implementation launches a certain number of software agents which negotiate which agents are responsible for execution of particular tasks. Secondly, the agents may run on a defined number of containers. This allows deployment and testing of both a fully centralized architecture in which one agent acts as a central management point or a fully distributed approach with several agents and containers. Obviously, the optimal number of agents and containers may depend on the size of the workflow as well as locations of actual services. Starting too many agents results in too much overhead for negotiation.

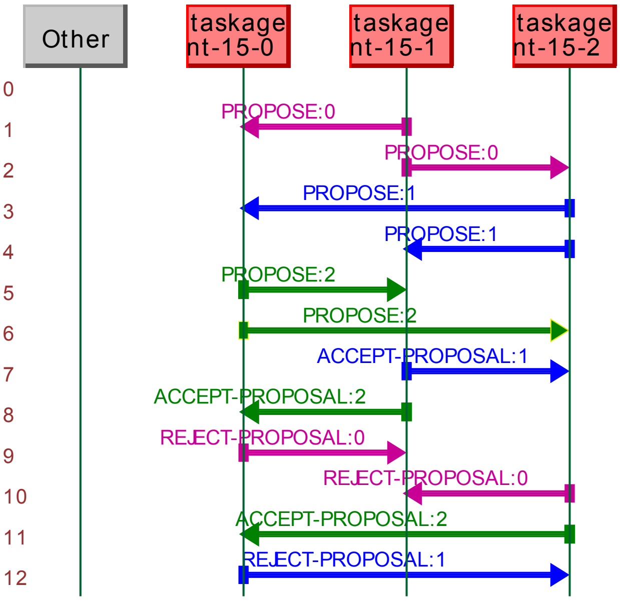

Figure 2. Negotiation between Agents

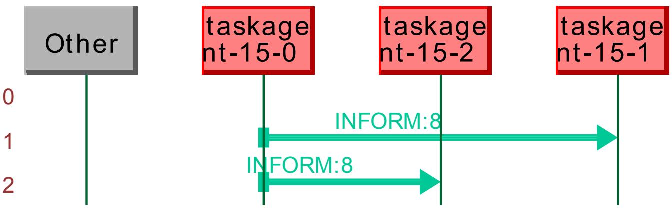

Figure 3. Notifications about Current Task Status

C. *Fault tolerance*

BeesyBees was designed with fault tolerance in mind. Currently, it is tolerant of service and TaskAgent failures. When either fails, the failure is recognised and either the respective flow path is restarted, or an alternative service is being sought and utilised. During task execution TaskAgent is blocking access to its assigned task by rejecting proposals concerning execution of that task. Thanks to this feature failure tolerance is achieved. When TaskAgent fails, nothing else is blocking an uncompleted task, so one of remaining agents picks it and proposes its execution to other agents. A service failure is detected after a specified number of unsuccessful retries. When this condition occurs an alternative service is chosen if available.

D. *Security*

At this point, we assume that the agent environment is managed by a group of trusted entities. We use security mechanisms implemented in JADE. Each container has its own certificate signed by our Certification Authority (CA) and it communicates with other containers using SSL.

Originally, BeesyCluster executes services on computing nodes using SSH. Each computing node has its own record

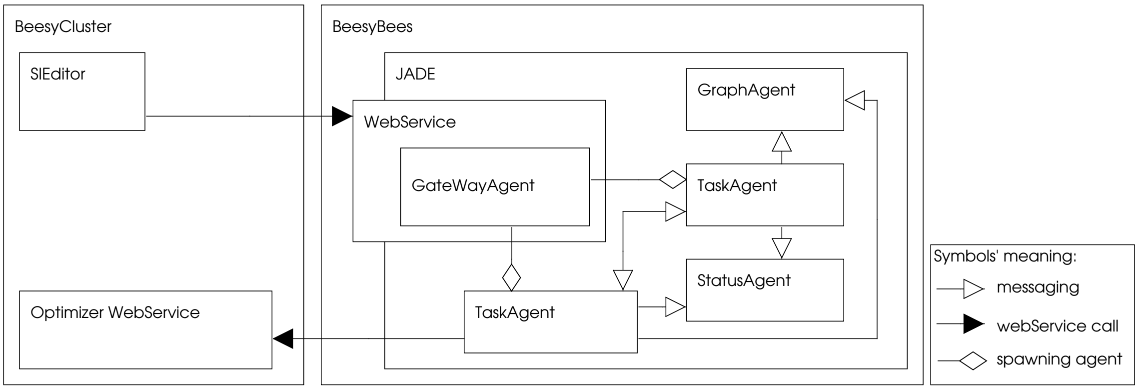

Figure 1. System Architecture

on the BeesyCluster's main server that contains its public key. In the agent-based approach we used a similar method. Each machine that contains an agent container has its own list of trusted computing nodes. Each agent can establish an SSH connection from a machine where it actually runs to communicate with computing nodes and invoke services.

The gateway agent is exposed as a Web Service and serves as a proxy between BeesyCluster and BeesyBees. The communication between BeesyCluster and the gateway agent is secured using HTTPS and certificates. There is always a possibility of attacks using holes that exist in software or solutions that we are depending on. In [26] such a problem as well as a way to mitigate its adverse effects is presented.

V. Other Agent-based Approaches to Workflow Management

One of similar projects we can mention is JBees [27] — a workflow management system based on agent technology combining collaboration agents and the coloured Petri net. As opposed to BeesyBees using BeesyCluster as a system for workflow creation, JBees makes use of built-in management agents providing user interface for the human workflow manager. Moreover resources in JBees, which can be compared to BeesyCluster services, are strictly integrated with the system and are represented by resource agents. The idea of process agents and storage agents is similar to BeesyBees task agents and state agents. Process agents are responsible for the execution of particular cases. Storage agents collect information from process agents (in JBees it is approached through the monitor agent, as opposed to BeesyBees where task agents communicate directly with the state agent) and make it persistent. As mentioned, resources in JBees are accessed by process agents requesting task execution through resource agents. In BeesyBees services are called directly by task agents. As for workflow description, JBees makes use of coloured Petri nets [28].

WS2JADE presented in [29] is a tool for runtime deployment and control of web services. It is of interest, because it uses the same agent system, JADE. The main goal of this project is integration of agent systems and web services, it is approached by representing each web service by a specialised agent. Similarly to JBees, web services are called through their associated agents. In order to call a web service, a client agent searches DF (Directory Facilitator, JADE built in service directory agent) for it, then if the service is not present, DF can trigger WS2JADE to look it up in the web service environment. If an appropriate service is found, the web service agent, which registers its service in DF, is created. Finally, the service is called by exchanging messages between agents.

SwinDeW-A [30] integrates services using WS2JADE by enhancing SwinDeW (Swinburne Decentralised Workflow) with agents. As opposed to BeesyBees using BeesyCluster's optimiser in order to choose best services for specified task, SwiNDew-A makes use of negotiation agents which are able to negotiate with a number of service agents. In order to choose the most suitable service, a Service Level Agreement (SLA) needs to be formed between the negotiation agent and each service provider. Agents can negotiate non functional parameters (such as time, cost, availability), which are similar to those proposed in BeesyCluster [3].

Finally, a relatively recent development is WADE (Workflow and Agents Development Environment) [31], which is an extension of JADE agent framework, facilitating both the possibility to define agent tasks according to the workflow metaphor and an architecture with mechanisms allowing administration of distributed WADE based applications. Workflows in WADE are expressed as Java classes.

VI. Simulations

A. Testbed Workflow and Environment

As an example, we have run a workflow application for parallel processing of RAW digital images in order to produce a Web album. The process uses standard steps in professional photo editing (`$1` is the input file name):

- `rawtotiff` — conversion from RAW to lossless TIFF. Implemented using the `dcraw` converter as `dcraw -T $1`

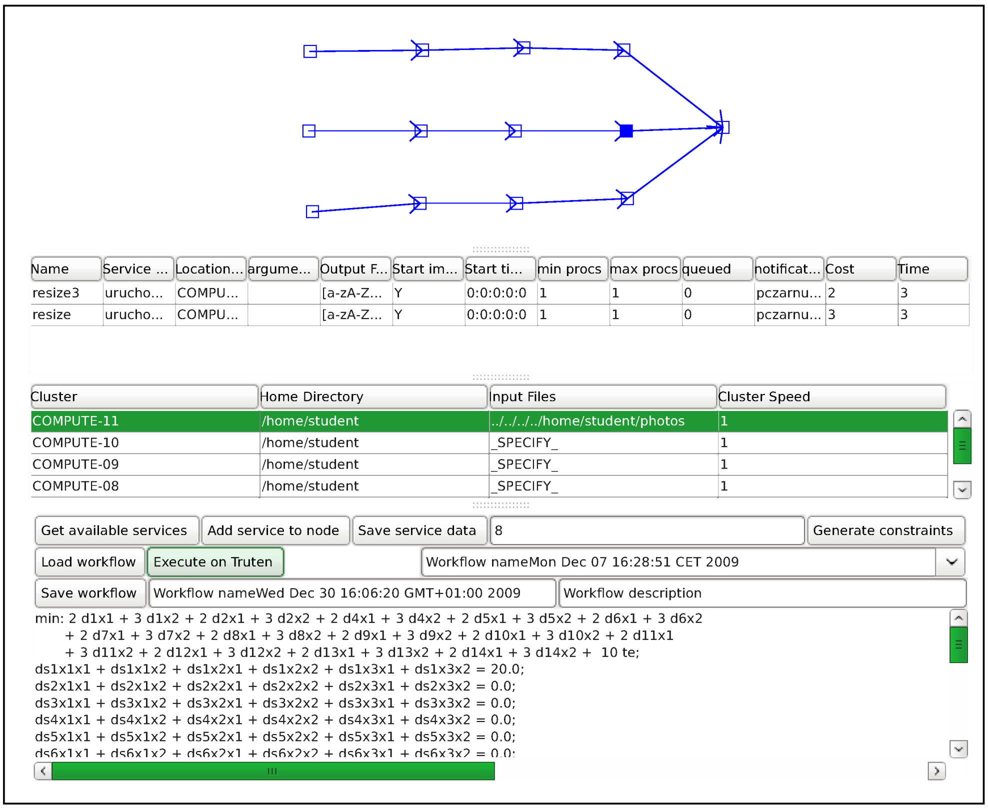

Figure 7. Testbed Workflow in BeesyCluster's Editor

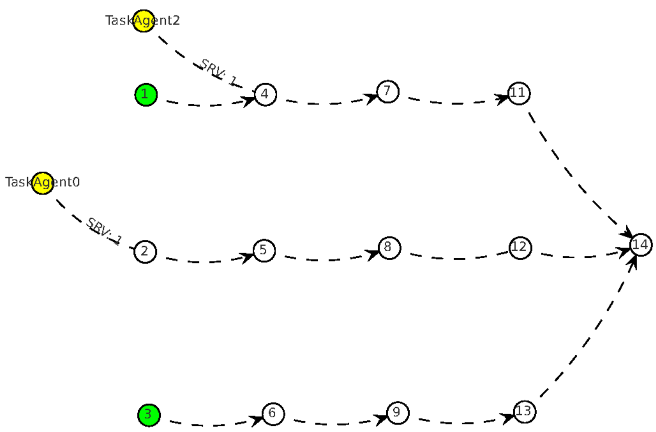

Figure 4. Visualisation of Workflow Execution

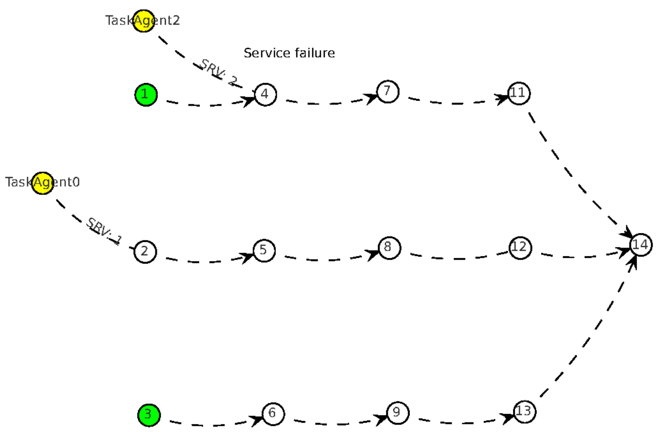

Figure 5. Visualisation of Workflow Execution with Service Failure

- `normalize` — normalisation of the image using ImageMagick's `convert` as `convert $1 -normalize $1`
- `sharpen` — sharpening the image implemented as `convert $1 -sharpen 1x1.2 $1.jpg`
- `resize` — resizing the image implemented as `convert $1 -resize 600x400 $1.jpg`
- `albumgeneration` — implemented by either `jigl` or `album`.

All those services are bash scripts executed through ssh. Input for each service contains pictures being processed.

Figure 7 presents the testbed workflow created in BeesyCluster's editor. Parallel paths include nodes running `rawtotiff`, `normalize`, `resize` and `sharpen` filters while the final node gathers resulting jpg images and produces

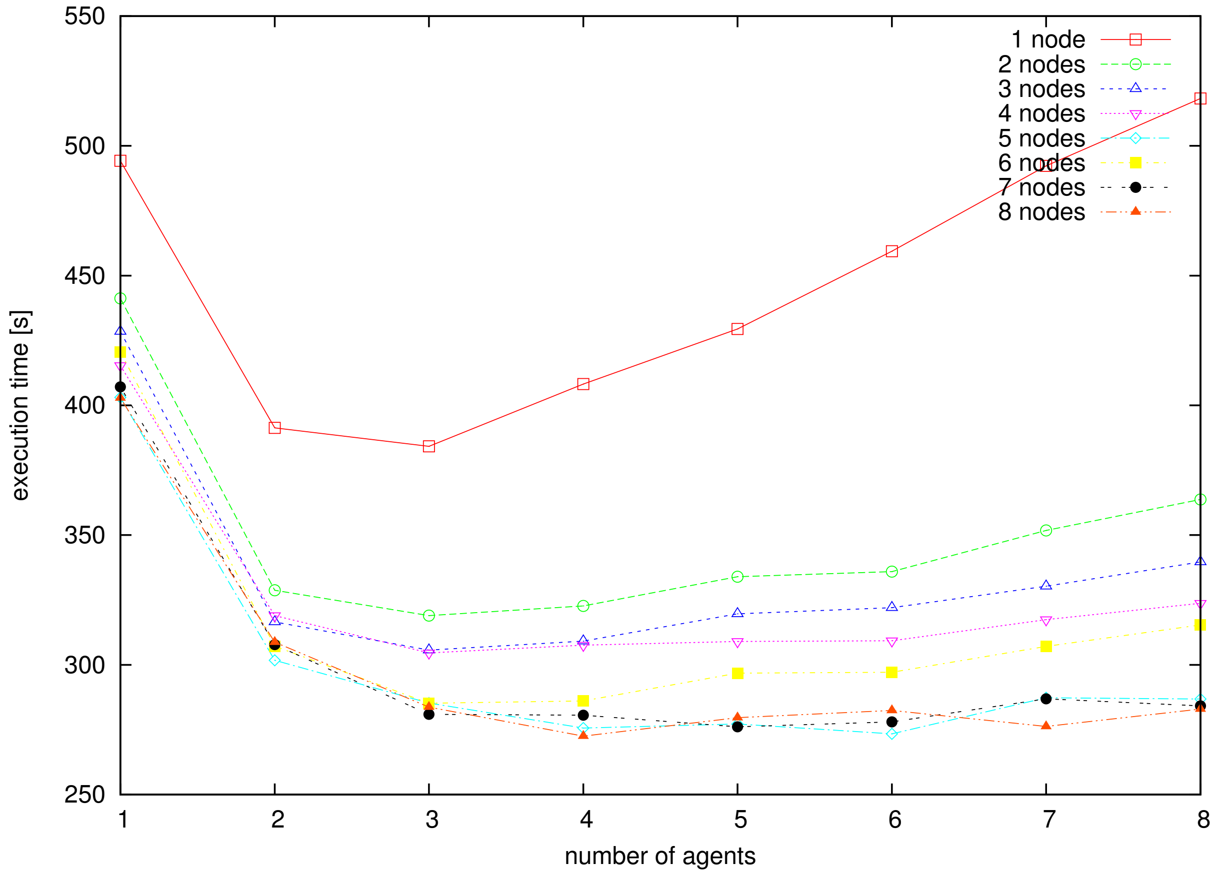

Figure 8. Execution Time

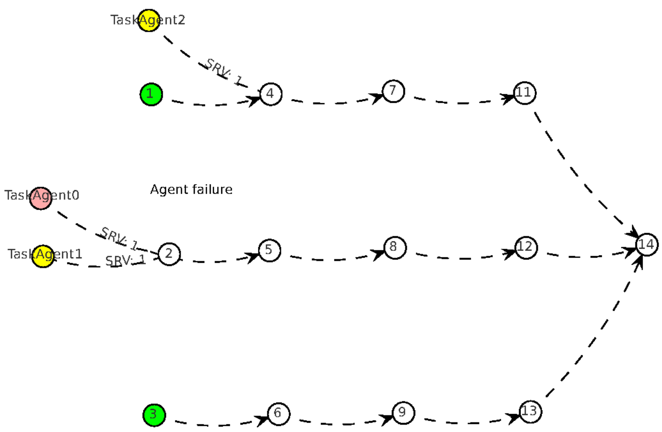

Figure 6. Visualisation of Workflow Execution with Agent Failure

a web album. For each node a primary and a backup services were deployed on a dedicated computer. Such a simple workflow was chosen on purpose, in order to facilitate the process of verification of experiment results with theoretical expectations.

There are several approaches to choosing appropriate services for tasks' execution. One is that following static optimisation done by BeesyCluster's optimiser before the run, task agents responsible for the given task call BeesyCluster's optimiser and fetch the selected service. Optimizer can consider such metrics like execution cost and time. Another approach, where knowledge about services is gathered during their execution and subsequently used to choose the best service considering metrics given, for example a probability of a service failure, was presented in [32]. If a particular service is not available or has a runtime failure, the agent selects the next best service for the task.

We have implemented a monitoring mechanism that shows the status of the workflow execution visually and updates it as the execution progresses. Figure 4 shows a snapshot of the testbed workflow being executed. Nodes 1–3 execute `rawtotiff`, nodes 4–6 execute `normalize`, nodes 7–9 execute `resize`, nodes 11–13 execute `sharpen` and node 14 executes `albumgeneration`. Figures 2 and 3 show communication between TaskAgents obtained by the JADE sniffing tool. They show the negotiation process described earlier and notifications about current task status sent to other TaskAgents respectively.

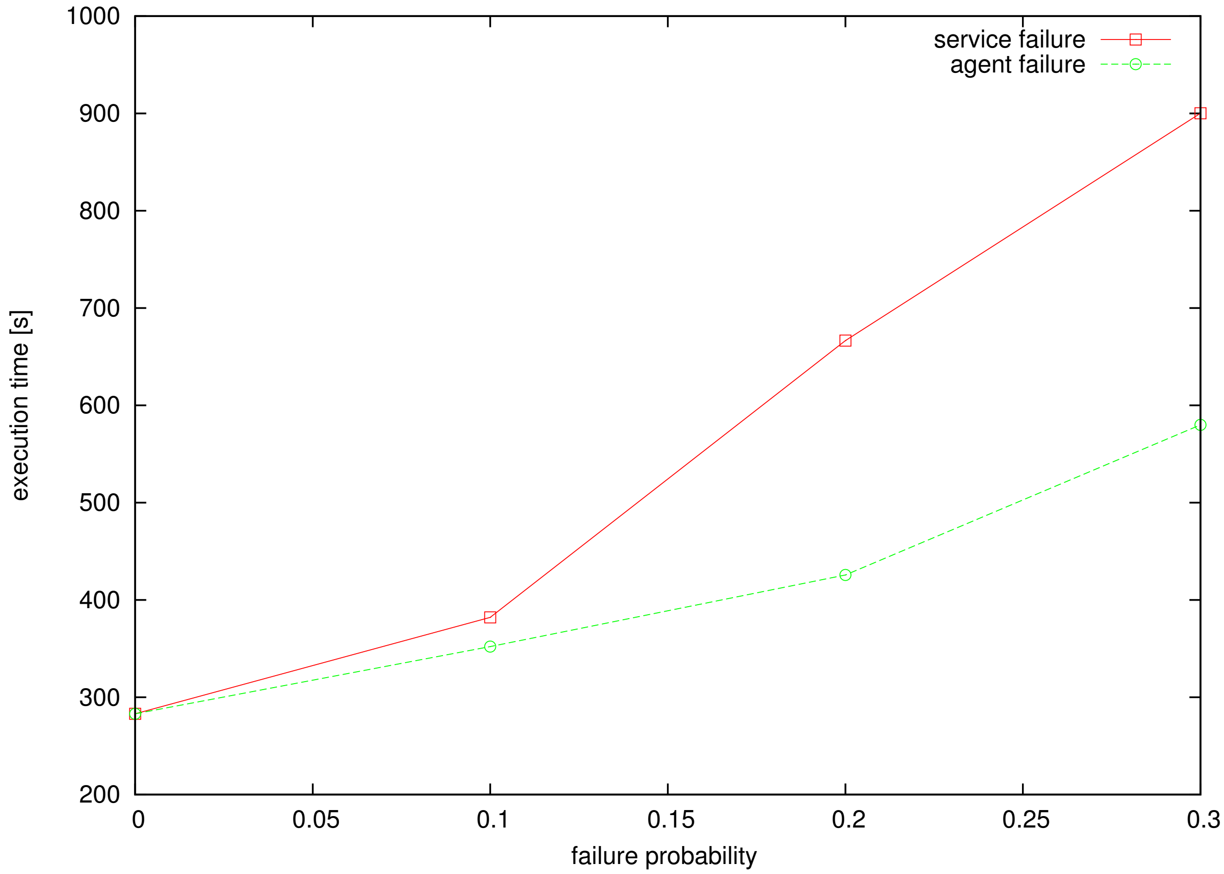

Figure 9. Execution Time in case of Failures

B. Optimization of Execution Time through Distributed Agent Processing

Figure 8 presents workflow execution time depending on the number of participating agents. They show execution times for processing the workflow with big files. There were 4 big files of 34MB each. Each of the series corresponds to a different number of system nodes on which agents work. The situation with only one agent and one agent node corresponds to a centralized environment, which performance should be similar to BeesyCluster's one because of using the same mechanism to invoke services. We can see that increasing the number of agents and nodes reduces the execution time, but only up to a certain number of agents. Further increase in a number of agents causes negotiation costs to offset any gains in execution time. In the testbed worklow there are three parallel paths and it can be seen that the results are best also for the number of working agents equal to three.

C. Mechanism for Fault-tolerant Execution and its Performance

We have been able to demonstrate that the execution environment is tolerant of faults in communication, service execution and agent execution. The workflow application can be completed successfully even if the following failures occur:

1) a service fails to complete after it has been invoked. As shown in Figure 5, if `taskagent2` detects a failure of service 1, it automatically switches to a functionally equivalent service 2 that executes the given task;
2) if a failure of a task agent occurs then another task agent takes over the responsibility of the former automatically. As shown in Figure 6, if `taskagent0` fails then `taskagent1` takes over its responsibility and the workflow execution can continue.

Figure 9 shows the total execution time of the worklow application assuming a certain probability of agent failure or service failure. It can be seen that execution time is getting higher with a higher probability of failure. If an agent fails, another one is given a chance for taking over the responsibility over its task and service. Again, it may fail with the given probability. For a given probability of agent or service failure, the final execution time is higher for the latter case. This is because for each task there is one agent but the service needs to be invoked four times because of four files being processed in each task. If any of the service invocations fails, a new service needs to be chosen, input files copied and again the service is invoked for the input files for the task.

VII. Conclusions

We described the architecture and its implementation that allows efficient, fault-tolerant and secure execution of workflows using agents. A standard execution management module would invoke remote services from one central server which would result in performance bottleneck and could fail if the server loses connections with the services. We have shown that for an optimal number of agents and containers, distributed management of workflow execution by agents is more efficient than a centralized approach. The distributed approach can complete successfully as long as agents supposedly located much closer to the services are able to reach the latter. We have been able to show that execution of a testbed workflow application for parallel processing of digital images completes successfully even if a task agent or a service fails.

Acknowledgment

Work sponsored by research grant N N516 383534 "Strategies for management of information services in distributed environments".

References

[1] J. Yu, R. Buyya, and C.-K. Tham, "Cost-based scheduling of workflow applications on utility grids," in *Proceedings of the 1st IEEE International Conference on e-Science and Grid Computing (e-Science 2005), IEEE CS Press*, Melbourne, Australia, December 2005.

[2] J. Yu and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms," *Scientific Programming Journal*, 2006, iSSN: 1058-9244, IOS Press, Amsterdam, The Netherlands.

[3] P. Czarnul, "A jee-based modelling and execution environment for workflow applications with just-in-time service selection," in *Proceeding of 4th International Workshop on Workflow Management (ICWM2009), 4th International Conference on Grid and Pervasive Computing, GPC 2009*, Geneva, Switzerland, May 2009.

[4] J. Yu and R. Buyya, "A taxonomy of workflow management systems for grid computing," *Journal of Grid Computing*, vol. 3, no. 3-4, pp. 171–200, September 2005. [Online]. Available: http://dx.doi.org/10.1007/s10723-005-9010-8

[5] M. Wieczorek, A. Hoheisel, and R. Prodan, "Towards a general model of the multi-criteria workflow scheduling on the grid," *Future Generation Comp. Syst.*, vol. 25, no. 3, pp. 237–256, 2009.

[6] B. Ludascher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, and Y. Zhao, "Scientific Workflow Management and the Kepler System," *Concurrency and Computation: Practice & Experience, Special Issue on Scientific Workflows*, 2005. [Online]. Available: http://www.sdsc.edu/%7Eludaesch/Paper/kepler-swf.pdf

[7] S. Majithia, M. S. Shields, I. J. Taylor, and I. Wang, "Triana: A Graphical Web Service Composition and Execution Toolkit," in *IEEE International Conference on Web Services (ICWS'04)*. IEEE Computer Society, 2004, pp. 512–524. [Online]. Available: http://www.trianacode.org/

[8] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny, "Pegasus : Mapping Scientific Workflows onto the Grid," in *Across Grids Conference*, Nicosia, Cyprus, 2004. [Online]. Available: http://pegasus.isis.edu

[9] *Parallel Grid Runtime and Application Development Environment, User's Manual, ver. 8.4.2*, Laboratory of Parallel and Distributed Systems, MTA SZTAKI, Hungary. [Online]. Available: http://www.lpds.sztaki.hu/∼smith/pgrade-manual/manual.html

[10] L. Young, S. McGough, S. Newhouse, and J. Darlington, "Scheduling architecture and algorithms within the iceni grid middleware," 2003. [Online]. Available: citeseer.ist.psu.edu/young03scheduling.html

[11] R. Aggarwal, K. Verma, J. Miller, and W. Milnor, "Constraint driven web service composition in meteor-s," in *Proceedings of IEEE International Conference on Services Computing (SCC'04)*, 2004, pp. 23–30.

[12] ——, "Dynamic web service composition in meteor-s," LSDIS Lab, Computer Science Dept., UGA, Technical Report, May 2004.

[13] Andrews, T., et al., "Business Process Execution Language for Web Services," 2003, version 1.1, BEA, IBM, Microsoft, SAP, Siebel.

[14] T. A. Engine, 2009, active Endpoints. [Online]. Available: http://www.activevos.com/community-open-source.php

[15] bexee, "BPEL Execution Engine," 2004, berne University of Applied Sciences. [Online]. Available: http://bexee.sourceforge.net/index.html

[16] G. Hackmann, M. Haitjema, C. Gill, and G. Roman, "Sliver: A bpel workflow process execution engine for mobile devices," in *in: Proceedings of 4th International Conference on Service Oriented Computing (ICSOC*. Springer Verlag, 2006, pp. 503–508.

[17] S. Ben Mokhtar, D. Fournier, N. Georgantas, and V. Issarny, "Context-Aware Service Composition in Pervasive Computing Environments," in *Rapid Integration of Software Engineering Techniques, Second International Workshop : RISE 2005*, Heraklion, Crete Greece, 2006, pp. 129–144. [Online]. Available: http://hal.archives-ouvertes.fr/inria-00415111/en/

[18] J. Han, E. Kim, and J. Choi, "Workflow language based on web services for autonomic services in ubiquitous computing," in *Proceedings of International Conference on Artificial Reality and Telexistence, ICAT*, Coex, Korea, 2004.

[19] P. Czarnul, "Integration of compute-intensive tasks into scientific workflows in beesycluster," in *Computational Science – ICCS 2006*, ser. LNCS, vol. 3993. Springer, 2006, pp. 944–947.

[20] "JADE (Java Agent DEvelopemnt Framework) Online Documentation," Telecom Italia Lab. [Online]. Available: http://jade.tilab.com/doc/index.html

[21] "FIPA specifications," The Fundation of Intelligent Physical Agents. [Online]. Available: http://www.fipa.org/

[22] "Mobile Agent System Interoperability Facilities Specification," Object Management Group. [Online]. Available: http://www.omg.org/cgi-bin/doc?orbos/97-10-05

[23] P. Czarnul, "Integration of compute-intensive tasks into scientific workflows in beesycluster," in *Proceedings of ICCS 2006 Conference,*. University of Reading, UK: Springer Verlag, May 2006, lecture Notes in Computer Science, LNCS 3993.

[24] P. Czarnul, M. Bajor, M. Fraczak, A. Banaszczyk, M. Fiszer, and K. Ramczykowska, "Remote task submission and publishing in beesycluster : Security and efficiency of web service interface," in *Proceedings of PPAM 2005*, Springer-Verlag, Ed., vol. in press in LNCS, Poznan, Poland, Sept. 2005.

[25] M. Matuszek, "Agent cooperation strategies in execution of complex distributed services," Ph.D. dissertation, Gdansk University of Technology, 2007.

[26] "Cve-2009-3555." [Online]. Available: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3555

[27] L. Ehrler, M. Fleurke, M. Purvis, and B. T. R. Savarimuthu, "Agent-based workflow management systems (wfmss)," *Agent-based workflow management systems (WfMSs)*, vol. 4, no. 1, pp. 5–23, 2006.

[28] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, Apr 1989.

[29] X. T. Nguyen, R. Kowalczyk, M. B. Chhetri, and A. Grant, "Ws2jade: A tool for run-time deployment and control of web services as jade agent services," *Software Agent-Based Applications, Platforms and Development Kits*, pp. 223–251, 2006.

[30] J. Yan, Y. Yang, R. Kowalczyk, and X. Nguyen, "A service workflow management framework based on peer-to-peer and agent technologies," in *Quality Software, 2005. (QSIC 2005). Fifth International Conference on*, Sept. 2005, pp. 373–380.

[31] "Workflows and Agents Development Environment," Telecom Italia Lab. [Online]. Available: http://jade.tilab.com/wade/

[32] P. Czarnul, M. Matuszek, M. Wójcik, and K. Zalewski, "Beesybees – agent-based, adaptive & learning workflow execution module for beesycluster," in *Faculty of ETI Annals, Information Technologies vol. 18*, 2010.