Proceedings of the ISCA 17th International Conference on Parallel and Distributed Computing Systems,
D.A. Bader, A.A. Khokhar (eds.), San Francisco, California, USA, September 15-17, 2004

An agent-based approach to support the scalability of change propagation

C. Constantinescu, S. Kornienko, O. Kornienko, U. Heinkel
Institute of Parallel and Distributed Systems
University of Stuttgart , Universitätsstr. 38, D-70569 Stuttgart, Germany

Abstract

In this paper, we address aspects of the challenge facing the Data Integration solutions in the problem of increasing the scalability. We first overview our results in data integration and present our prototype. Searching for a motivation to employ the agent-based technology and envision how this technology can be applied to improve the system scalability represent the next step of our research. For that, we examine some formal definitions and metrics of scalability which fit the purpose of system description. The analysis of our data integration solution, a change propagation system called Champagne, identifies the system components having the most influence on the scalability problem. We propose to employ the agent-based technology to improve the scalability and perform some experiments which reveal our work.

1 Introduction

Most enterprises have a diverse environment of heterogeneous and autonomous information systems. If the same data is relevant for several information systems, then data changes in one system affect data stored in other systems. The integration of data as well as of functionality is generally termed *enterprise application integration* (EAI) [3]. The management of a single, integrated enterprise information system is often infeasible or too expensive, due to the autonomy of information systems and the heterogeneity of their IT infrastructures.

The solution is to support the enterprise by a generic approach able to manage data dependencies and to transform data stored in a source information system according the specifications of the dependent information systems [2]. The solution has to possess specific features that enable the operation in *turbulent* and *unpredictable* environment. It has to face an increasing number of systems which intend to benefit of the service of change propagation. These systems are characterized by high diversity and the heterogeneity of their data structures. At the same time the approach has to process a growing flow of changed data between already connected systems. These challenges bring the problematic into the field of scalability. Moreover, the operation in a turbulent and unpredictable environment demands the ability to monitor, react and rapidly adapt to these changes. This capability supports robustness, reliability and availability of the change propagation solution.

We propose to improve scalability by employing the results from the field of cooperative information systems. The agent-based technology has proved his "collective intelligence" to support cooperation and communication in complex processes [7]. However this technology has its own price, consisting of hard modelling and simulation work before its implementation. Based on these assumptions we investigate through several experiments the efficiency and effectiveness, as well as the general benefits of using agent-based approach in the problem of change propagation.

The paper is structured as follows. Section 2 gives an overview of our results and implemented prototype in the field of data integration. In Section 3 we identify a motivation to employ the agent-based technology and envision how this technology can be applied to improve the system scalability. Then, in Section 4, we propose to use the agent-based technology to improve the scalability of our solution and perform some experiments. Finally, in Conclusions, we analyze the obtained results and conclude about expectation of applying the agent-based technology in the change propagation solution.

2 Champagne, a solution for the change propagation

This section overviews the architecture, the main components and the features of our approach in the

field of *enterprise application integration*.

We developed a data change propagation system called *Champagne* (change propagation manager)focused on data integration level of EAI [3]. Our prototype consists, of the following main components, the *propagation manager*, the *dependency manager*, and the *repository*. The detailed flow during the processing of change propagation between a source system, and the dependent system is presented in [2].

Our implementation is based on several concepts introduced in [2]. We recall the definitions of the main used terms in this paper.

Change propagation is the process of forwarding a data change from a source system to all dependent systems. We use the term *dependency* to describe a directed relationship between a source and several destination systems. It specifies the source, the destination systems and a *propagation script*. The propagation script contains information required for transformations (defined by *transformation scripts*), filtering, and routing. The Propagation Manager transforms, filters, and routes changed data using the dependencies that have been created, stored, and updated by the Dependency Manager. Its main component, the Propagation Engine, interprets the scripts based on a workflow specification language. The Repository stores the dependencies, the XML schema descriptions of the systems, and propagation and transformation scripts. The Adaptors map data between the local schemas of the connected systems and the corresponding XML schemas. To enable both synchronous and asynchronous communication between the adaptors and the propagation manager, we use *Java Message Service* (JMS).

The combination of XML technologies and the explicit treatment of dependencies via a modular design guarantees the necessary functionality and flexibility of change propagation in EAI. For the sake of platform independence, a vital requirement in heterogeneous environments, we built all components in Java. We developed our prototype as part of a larger research project on innovative concepts and techniques to enable highly flexible series production systems in the manufacturing industry [1].

Champagne provides a high degree of *reliability* and *availability*. For example, failures in participating systems are handled in a way that other systems are unaffected. Finally, Champagne has to be *scalable*, i.e., the resources (time and memory) required to manage dependencies should grow (linearly) in proportion to the number of dependencies and to the amount and size of data to be transformed and propagated as well. The scalability represents at the moment one of the challenge we have to face.

[1]SFB 467 "Transformable Business Structures for Multiple-Variant Series Production", funded by the the German Research Foundation.

3 Motivation and vision for improving scalability in change propagation

We envisioned in [2] as our future work the employment of agent-based technology in the problem of data change propagation. We arrived now to the point to search for a motivation of using this technology for the purpose of improving our system. While a challenge to confront our Champagne is the scalability, we examine in the following some possible ways to address it. First, we give an overview of the scalability by its definitions, typology and metrics. Then, we discuss our system from the scalability point of view and identify its components which play critical role in scalability increasing. Based on our results in the field of MAS [5], we envision several agent-based mechanisms supporting the solution of our problem.

3.1 Phenomenon of scalability in collective information systems

Scalability represents a desirable feature of systems, meaning economical functioning in a wide range of sizes and configurations. Several authors often define scalability in terms of productivity and performance [4]. In the domain of multi-agent systems scalability can be also stated in terms of co-ordination policies as e.g. total number of message exchanges necessary to converge on a solution [7]. In the field of enterprise application integration the scalability refers to the challenge of making the integration solution to achieve the propagation of changes across the growing number of connected applications or information systems.

Type	Description
Load scalability	functioning at different loads
Structural scalability	growing number of subsystems
Diversity Scalability	growing heterogeneity degree
Dynamic Scalability	shifting to short-term dynamics

Table 1: *Different types of scalability.*

We show in Table 1 the identified types of scalability relevant for our work (see e.g. [1]). Returning to collective information systems, we remark that

scalability differs here in several ways from other, like multiprocessor, systems. Firstly, scalability in collective information systems means primarily structural scalability of two types: growing amount of identical components as well as growing diversity of components, i.e. *structural and diversity scalability*. Almost all collective systems reveal this property. Secondly, at growing number of components, the load on the system increases as well. Therefore *load and dynamic scalability* are also typical for collective information systems. There are several challenges we have to face in each case of above mentioned scalability types. The *structural scalability* induces the increase of coordination effort to solve a problem, the overload of communication, the explosion of communication costs and the fail of communication channels. The *diversity scalability* has impact on the flexibility with the need of adaptations for the new components and on the functional adaptation of collective procedures to new components. The *load and dynamic scalability* influences the appearance of overloads and bottle necks, the growing of probability for partial failure's and the need of activities monitoring.

In the following we employ agents-based technologies to solve problems arising primarily at *diversity scalability* as well as *load and dynamic scalability* in collective information system. Our case study refers the data change propagation system Champagne, briefly presented in the previous section. This field of application allows without loss of generality to consider the common tendencies of providing especially structural scalability and illustrate them by a practical example.

3.2 The challenge of scalability in change propagation

In our usage, scalability refers the challenge of making the propagation system to achieve its role across the number of connected information systems.

To propose a solution to this problem, we analyze the case of an increasing number of connected systems which take benefit of change propagation service offered by Champagne. Suppose that a new system intends to propagate across all interested systems its changed data, through our approach.

First, Champagne has to manage (create, store, use) in the repository all data regarding the new client-system: system name and its associated XML schemas, authentication information needed when system connect to Champagne, all existing dependencies, consisting of the name of the source (here the new system) and destination systems and their schemas, as well as the name of the propagation script, propagation scripts involved in any dependency, and transformation scripts referenced in any propagation script.

Second, Champagne has to deliver to each new system an adaptor which provides the specific bidirectional translation between its local data representation and a representation that conforms to an XML schema. The designer of the adaptor has to define an appropriate schema in the repository using the schema editor of the dependency manager. Thus, we have identified two critical components involved in connecting a new system and responsible for the system scalability: the dependency manager and the adaptors. As a conclusion, there are at least these two directions which motivate our work to support the scalability of change propagation with agent-based technology.

3.3 Envisioned mechanisms providing scalability

As pointed out in the previous sections, the problem of scalability is closely related to the changes of system's relevant parameters and to the reaction of the system on these changes. In this context we distinguish between *scaling values* (parameters) and *absorbing values*, in charge of absorbing the changes caused by scaling. The scaling mechanisms have to provide the stable working with similar performance in some range of scaling as well as absorbing parameters. In the case of Champagne, we consider as one of the relevant parameter the number of connected systems. If a new system is coupled, the number of connected systems changes. The reaction of Champagne to this change consists of the creation and storing all information related to the new system, in the repository: names, XML schema, dependencies and related propagation and transformation scripts.

The use of mechanisms proving scalability brings us in following three cases of interest: the system is nonsensitive to the variation of scaling values; the changes of scaling parameters can be absorbed by similar changes of absorbing parameters in linear or sublinear proportion; and the scaling changes require multiple modifications of absorbing parameters or even modification of system's structure.

The first case is the most relevant for the scale-invariant systems. Although this issue is of huge interest, the treatment of this problem oversteps the framework of this work.

The second case represents the standard speedup solution: a growth of scaling parameters is compensated by equivalent growth of resources. For example, each new change propagation query in Champagne is

connected with a new handling process. Thus, N new queries create in parallel exactly N new processes, the total system's performance remains the same for different load N. If we can guarantee that the consumed resources by these processes growth sublinearly or linearly with N, this scheme can underlay the mechanisms providing scalability. However, in many cases, the speedup schemes cannot assure a complete absorbance of changes.

The reasons of this deficiency consists in a nonlinear multiple dependence between scaling and absorbing values. In this last case we cannot absorb the changes in the speedup way. As a result, we need to perform more complex modifications, even to modify the structure of the system. For example, increasing the frequency f_q of querying, we can achieve the limit, where new processes get started when already started processes are not yet finished. These N new handling processes will be added to the not-finished ones and the total number of started process N_p growths exponentially with f_q. The speedup solution is not useful here, this problem can be solved only by shortening the time required by handling process. This, in turn, requires multiple modifications of the system structure.

The speedup and the multiple modifications represent two main scaling mechanisms utilized further. We propose to employ the Multi-Agent-Systems (MAS) for implementing these mechanisms, based on two reasons. *The first reason lies in the autonomy of agent behavior.* Each agent executes cyclically a sequence of activities, like collecting of information, planning and so on. These activities can include monitoring of scaling values to perform an adaptive speedup. An agent can continuously monitor changes in the system in order to start new propagation processes, as well. Generally, the autonomy gives to agent-based approaches the ability to react dynamically in continuously changing environment.

The second reason consists in agent's ability to find collective solution based on negotiation. The point is that the collective solution space is essentially larger, than that of individual agents. The agents group can find (and optimize) the solution in situations where separate agents fail. This property is of essence for the mechanisms involving multiple modifications. In the performed experiments the dependence between some scaling and absorbing value continuously changes. It is very difficult to find an optimal relation (and sometimes even a non-optimal one) in advance. Therefore agents, based on negotiations, have to find this relation so that to guarantee scalability in a given range.

4 Multi-agent change propagation system

4.1 Multi-agent architecture of dependency manager

We propose a MAS architecture of the dependency manager in order to implement scaling. The architecture consists of two layers, focusing on *primary* and *secondary* activities, as presented in Fig. 1. The *primary activity* layer implements the constraint-based matching that underlies the dependency manager during the process of dependency creation. The **Input Guard** agent monitors the changes in a given system and triggered by a changed data starts the **Process Agent**. This has a hierarchical structure and consists of the **XGT** (**X**ML-**G**raph-**T**ransformer), **Matching**, **Constraint Conformer** and **Script Generator** agents. XGT agent gets two XML scripts from the Input Guard, describing the data structures of source and destination systems. These XML structures are converted into an internal graph representation. The Matching Agent receives these graphs and performs the matching. The contained XML constraints are processed by the Constraint Conformer agent. The goal of this operation is to find compromises between constraints of source data structures and destination data structures. Finally, the Script Generator agent creates the propagation scripts. The details of agents implementation as well as of the performed operation are given in the next section.

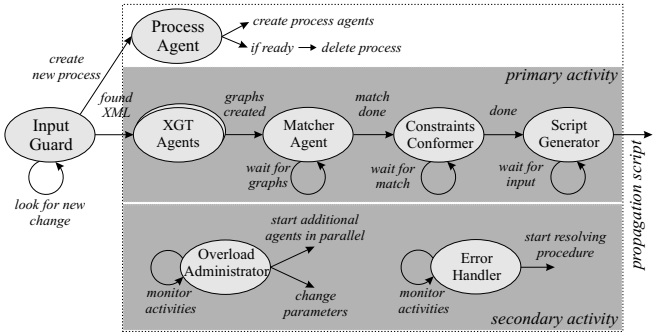


Figure 1: *Structure and main activities of multi-agent dependency manager.*

The scaling values of this architecture are represented by the frequency of querying f_q, the number of querying processes at once N_q and the diversity D_{XML} of XML data structures. These values correspond to the load and diversity scaling, mentioned in Section 3.1. The absorbing values are the number of

simultaneously started processes N_p and the number of started Matching Agents N_m. In the speedup solution we create constant linear dependency between N_q and N_p and variable linear dependency between D_{XML} and N_m. The creation of nonlinear dependency between absorbing values N_m, N_p and the frequency of querying f_q is used in the mechanism of multiple modification. The details as well as results of experiments are discussed in Section 5.

The *secondary activity* layer handles the irregularities arising during the constraint-based matching. We identify two kinds of irregularities: overloads and errors in XML data structures. These are managed correspondingly by the **Overload Administrator** and the **Error Handler** agents. Since Error Handler does not affect the scaling properties of the propagation system, its activities are limited by simple generation of error messages. However overloads are closely connected with scaling values (they cause overloads). To handle them we utilize the collective properties of multi-agent system to find such a combination of system's parameters to absorb the overload.

For the Overload Administrator agent we apply a simple evolutionary strategy of collective solution as presented in Table 2. The Overload Administrator agent monitors system's load (number of started agents, resources consumed by agents) and, at overstepping some threshold, it calls other agents and suggests them to change the absorbing values that these agents have available. The called agents start the role *Change Absorbing Value* and try to vary the parameters. They continue the changing of parameters if the load goes down, otherwise they start to try new combination of parameters. Since the number of possible absorbing parameters in the dependency manager is small, agents do not need to synchronize explicitly the changing behavior. The changes of load allow implicit synchronization. Although this simple strategy cannot guarantee convergency for a large number of agents, for a small number of agents it allows finding the gradient and continuous descent on the gradient.

agent:role(Overload Administrator)
  do always monitor system's *load*
  activate if *load* > *threshold* **do iteratively**
    create (*list* of agents)
    call agents (from *list*) **set role**=change absorbing value
  finish iteration if *load*< *threshold*
  endactivate
endrole

agent: extend agent (*name*)
    :role(Change Absorbing Value**)**
  create (*list* of absorbing values)
  do iteratively (change *values*)
    synchronize agent (empty)
    if *load* goes down **do** further
    finish iteration if *list* empty || *load* goes up
endrole

Table 2: *Example of role-based evolutionary strategy for collective solution finding.*

4.2 Details of modelling and performed experiments

We implemented the agent-based dependency manager in Java, by using the AnyLogic[2] simulation engine for managing agent activities. Instead of converting XML data structures, we create in our experiments the graph according the following format:

*Field*1 {*Subfield*1 (value, constraints),
  *Subfield*2 (value, constraints),...},
*Field*2 {*Subfield*1 (value, constraints),
  *Subfield*2 (value, constraints),...},
...

The number of fields and subfields is random between 1 and *FieldMax*=[1 - 1000]. For matching, we use the approach suggested in [6], which recommends the matching of *name similarity*, *position in hierarchy* and *similarity of constraints*. The constraints are given in the form of *value type* (integer, double, string) and *permissible range* (e.g. 0-1000). In the match of candidates with similar constraints (e.g. integer → integer, or integer → double) we perform transformation of constraints. The Input Guard agent can be activated 1-100 times per second (f_q) and it can start 1-1000 process agents simultaneously (N_q). The number of matching agents N_m varies between 1 per process and *number of Field* (we perform parallel matching on the subfield level only, since deeper level of parallelization brought no added benefit). The experiments are performed with a single-processor machine Pentium 2,0 Mhz with 512MB memory. Since we use a single-processor machine, the started agents are executed by the AnyLogic simulation engine in sequence. Therefore the number of simulation steps, required for *one agent* to perform all activities, can be assumed to be proportional to consumed system's resources (CPU time, memory and so on) by the whole simulation.

[2]www.xjtek.com

This value is adopted as a consumption of system's resources.

5 Discussion of results

We mainly use in our experiments the load as the *number of querying processes started at once* N_q and the *frequency of query to propagate changed data,* f_q. Figs. 3(a) represent the load of the system through these two parameters. Fig. 2 highlights that the system is Superscalable at the growing number of parallel processes. This points to the coincidence of the results with the speedup scheme. The consumed system's resources growth linear (in this case slightly sublinear) with the number of processes. In a multi-processor implementation the consumed resources by one process are expected to remain constant.

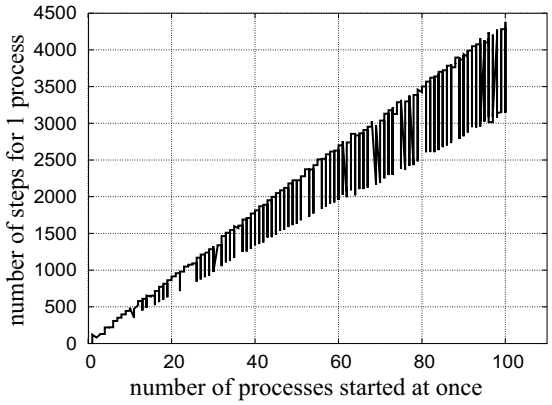


Figure 2: *Dependency between the load and consumption of system's resources.*

According to the experiments shown in Fig. 3, the system has for the variable f_q the range [0 - 8], where the number of processes remains constant. However beyond this range the number of processes (3(a)) as well as the consumed resources (3(b)) growth exponentially with f_q. As already mentioned, we explain this behavior through new processes which get started when already started processes are not yet finished. The size of queue containing not-finished processes is constant for the same f_q, but growths exponentially with f_q. The size of queue depends on several other parameters, the most important is the length of process. This depends on matching parameters, on the number of matching agents, on the constraints converting procedure and so on. After discussing the meaning of these parameters for diversity scaling, we return to the frequency of querying, in trying to improve this relation.

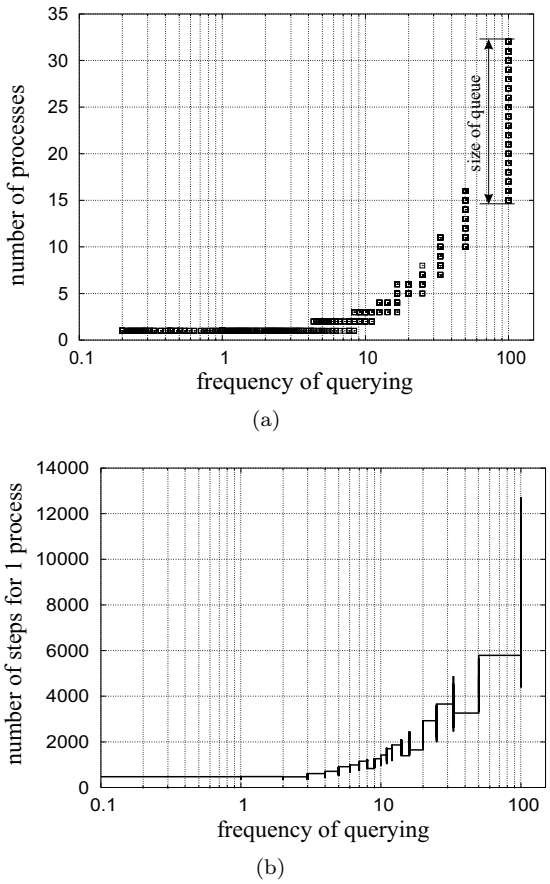


Figure 3: **(a)** *Load scalability as the frequency of querying,* $N_m = 1$*;* **(b)** *Dependency between this load and consumption of system's resources. Both dependencies are shown in logarithmic scale of the x-axis. Overload Administrator is off.*

5.1 Diversity scalability

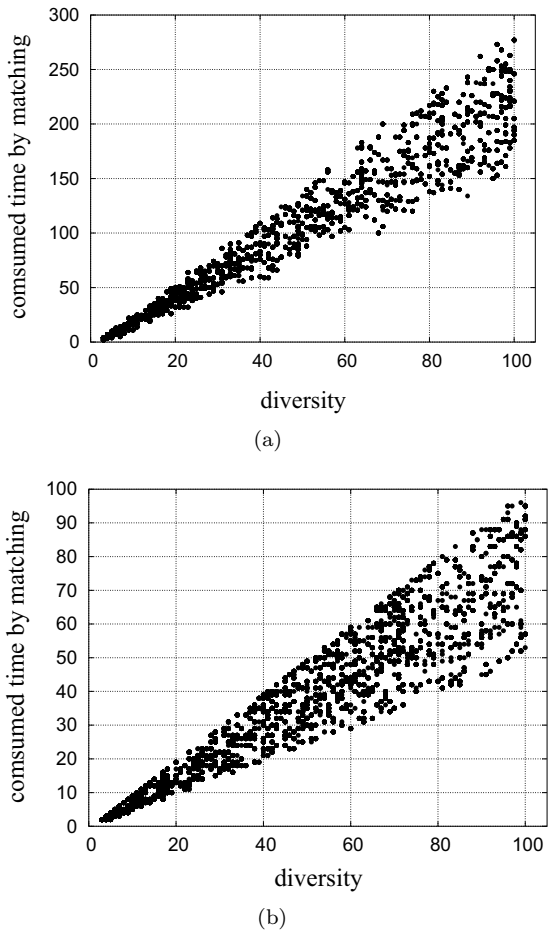
We define and use the term of diversity scalability as the number of fields that do not coincide in source and destination data structures. The dependency between the length of matching and the maximal number of fields is reflected in our approach by defining the following parameter:

$$K = random(\frac{maxFields}{100} maxDiversity, maxFields) \quad (1)$$

The difference between values of K_1 (calculated for the first graph) and K_2 (for the second graph) gives us the real diversity. The values of *maxDiversity* (expressed in %) describes how the first graph differs from the second one (for $maxDiversity = 100\%$ they do nor differs at all). The *maxFields* is the maximal length of fields in the relation (1). To demonstrate the de-

pendency between length of fields, diversity and the time of matching we adopt *maxFields* as *the value of diversity* in the performed experiments.

We illustrate in Fig. 4 the dependence between *maxFields*, in the range [3 - 100], and the time needed for matching. The matching time is taken as the number of matched fields multiplied on the number of steps required to match one field. Fig. 4(a) shows this dependence in the case of one matching agent started, whereas Fig. 4(b) presents the case where the number of started matching agents equals to the number of *Fields* in the graphs. The linear dependency between the number of matching agents working in parallel and the corresponding consumed resources is presented in Fig. 5.



(a)

(b)

Figure 4: **(a)** *Diversity scalability (*maxFields*) at* $N_m = 1$, $f_q = 1$*;* **(b)** *Diversity scalability (*maxLFields*) at* $N_m =$ *number of Fields,* $f_q = 1$.

A comparison between Figs. 4 and 5 reveals that the load scalability (frequency of querying) has greater impact on Champagne than the diversity scalability.

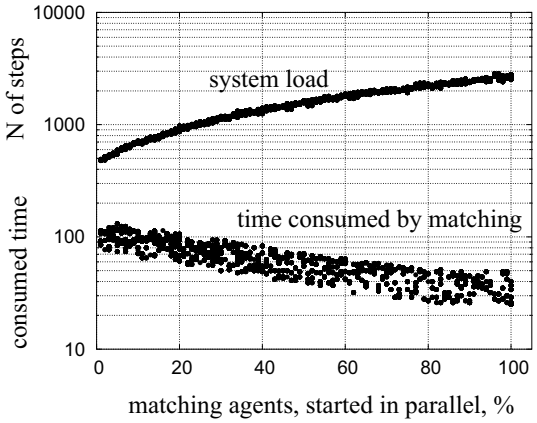


Figure 5: *Dependence between the number of matching agents, working in parallel, resources consumed by agents and the time needed for matching.*

The resources consumed by agents growth linear (like in Fig. 2 slightly sublinear) with the number of agents that points to a constant consumption in the multiprocessor implementation. The time of matching is closely related to the size of the queue in the case of increasing frequency of querying. We discuss this dependence in the next section.

5.2 Improving load scalability

To improve the load scalability, we intend to short the process time. In the proposed architecture, the most time is consumed by the matching agent. Therefore, shortening the matching time, we can reduce the process time. Two parameters enables us to reduce the matching time: the number of simultaneously started matching agents N_m and the way how to perform the distributed matching (e.g. the level of distribution). Changing the level of distribution from *Fields* to *SubFields* or even deeper to *Value, Constraints*, in the performed tests, we did not achieve a real increase of performance, because to collect the matched results from the deepest level agents consumes time, as well. This collecting time finally growths proportional to the level of distribution. Therefore N_m remains the only effective parameter that can be handled.

The Overload Administrator agent monitors the system's load (in this case the number of processes N_p in queue) and if N_p growths considerably it calls the role *Change Absorbing Value* of other agents. Since N_m is only one effective parameter in the system, the call of this role changes only the number of simultaneously started matching agents. The dependency between the number of started processes, the consumed

resources by agents and the frequency of querying is shown in Fig. 6.

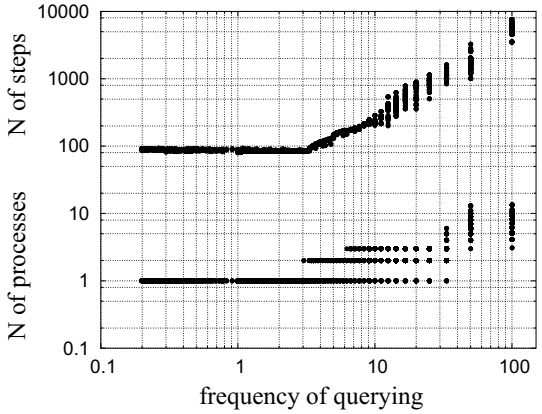


Figure 6: *Dependency between the number of processes, consumption of system's resources and frequency of querying if the Overload Administrator is on. Both dependencies are shown in logarithmic scale of the x,y-axis.*

Comparing Figs. 3 and 6, we conclude that the number of processes in the queue do not growth significantly if the Overload Administrator is on. However we are not able to achieve the constant size of queue. Moreover, the consumption of system's resources still increases exponentially with f_q. We give to this result the following two explanations. Firstly, there is a "physical" limit imposed on the minimal time of processes. If we increase the querying frequency so that it oversteps this limit, there is no approach that can avoid increasing the queue size. Secondly, the mentioned "physical" limit depends on several parameters. Modifying these parameters, we could, in principle, reduce this limit. However in the used simple architecture there is no enough degrees of freedom (parameters) to achieve it. Agents can modify only the number of started matching agents. This strategy leads finally to exponential consumption of system's resources. In the architectures, possessing more degrees of freedom, the more "intelligent" solutions towards improving load scalability are expected to be found.

6 Conclusions and future work

In order to employ agent-based technology in the field of change propagation we performed some experiments which proved the expected benefits of our proposal. This work represents crucial steps before implementing the first prototype of our agent-based approach. As shown by experiments, the highest impact on the scalability represents the system's load. The load consists of the number of simultaneously started querying processes N_q, in our case propagated changed data, and the frequency of querying f_q for propagation. The proposed agent-based solution supports large scaling of N_q-load and restricted scaling (in the given range) in the case of f_q-load. We identified that the f_q-load is of "physical" nature and this cannot be got round. The consumed resources depend linearly on the number of started processes (in area where f_q-load is constant). We conclude that the real implementation of such an agent-based system is feasible.

References

[1] A.B. Bondi. Characteristics of scalability and their impact on performance. In *Proc. of the second international workshop on Software and performance*, pages 195–203. ACM Press, 2000.

[2] C. Constantinescu, U. Heinkel, R. Rantzau, and B. Mitschang. System for data change propagation in heterogeneous information systems. In M. Piattini, J. Filipe, and J. Braz, editors, *Enterprise Information Systems IV*, pages 51–59. Kluwer Academic Publishers, 2003.

[3] A.D. Jhingran, N. Mattos, and H. Pirahesh. Information integration: A research agenda. *IBM Systems Journal*, 41(4):555 – 562, 2002.

[4] P. Jogalekar and M. Woodside. Evaluating the scalability of distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 11(6):589 – 603, 2000.

[5] S. Kornienko, O. Kornienko, and J. Priese. Application of multi-agent planning to the assignment problem. *Computers in Industry*, 54(3):273–290, 2004.

[6] E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal: Very Large Data Bases*, 10(4):334–350, 2001.

[7] O.F. Rana and K. Stout. What is scalability in multi-agent systems? In *Proc. of the fourth international conference on Autonomous agents*, pages 56–63. ACM Press, 2000.