Services Allocation Planner for Dynamic Reconfiguration

Bogumil Zieba

Thales Nederland B.V., Haaksbergerstraat 49, Hengelo (O) 7550 GD, The Netherlands
bogumil.zieba@nl.thalesgroup.com
http://www.thales-nederland.nl

Abstract

The system consists of collection of services, running on several of nodes. The workload generated by services may cause nodes overload, which negatively impacts the response time of services. The reconfiguration of the system incrementally transforms one system state to the other. The goal of the reconfiguration is to transform the system state into one, which all services meet their response time. Finding the appropriate reconfiguration that satisfies all timing requirements is the dynamic, preemptive scheduling problem. We consider this problem as the planning problem, where plan consists of consequent steps of the reconfigurations actions. Classical planning algorithms cannot be applied to this problem because of the incomplete information. We used the concept of the conformant planning (Smith and Weld 98), which refers to planning with incomplete information that achieves the goal from any initial state compatible with the available information. The planner consists of the plan generator and the plan selector. The generator creates a collection of plans. The selector selects that one, which in the most probable degree transforms system to the state that satisfies all timing requirements. The selected plan is applied to the system. The simulation results proved the concept of our approach.

Introduction

The context of the research is distributed, mission-critical systems, where the functioning of an organization or success of a carried mission depends on the predictable and reliable system operation. National interests are becoming increasingly dependent on the continuous, proper functioning of large-scale, heterogeneous, and decentralized computing enterprises. Examples of such systems abound, ranging from military command and control to vital national security assets such as the financial and banking system (Wolfy et al.). The critical character of such applications introduces high quality requirements (ultra-quality (Rechtin and Maier 97) for these systems. The specification on the mission-critical systems requires that the systems behave correctly i.e. exhibits desired functionality in all circumstances (Muhl 02). This can be achieved by providing systems with mechanisms that protect it before the occurrences of events that could have impact negatively on system behavior e.g. in terms of system performance. Proactive reconfiguration adds, removes, and replaces components and interconnections to cause a system to assume postures that achieve enterprise-wide intrusion tolerance goals, such as increased resilience to specific kinds of attacks or increased preparedness for recovery from specific kinds of failures (Wolfy et al.). The example of this kind of mechanism is presented in (Zieba et al. 05).

Computational model

The problem statement of the research is very much dependent on the assumed computational model. It consists of distributed services that are run on the computing nodes. One service is run on one node. The location where service is run can be changed at runtime. The crucial requirement for this computational model is the time-constraints. It states that the system shall execute a collection of transactions in such a way that all time-critical constraints meet their specified deadline. The service needs computational, and communication resources to execute. These needs are specified as the required QoS (Quality of Service), which are defined during the design phase by a service-developer. This concept is based on the statement that not all services need the same performance from the resources over which they run. Thus, services may indicate their specific requirements to the resources, before they actually start transmitting data (Fluckiger 95). The service-developer specifying required QoS ensures that the service always meets its response time. This time is defined as the time that elapses between service invocation/read data and completion of the service request/write.

The context of this research is the distributed system that exhibits high-level of the dynamism. The computing nodes and services may leave and join at the arbitrary moments. The system configuration needs to be adaptable to cope with changes in the environment and in the computing platform.

These dynamic situations are subject of uncertain services behavior. For example, in the case of a dynamic reduction of available resources, services may not meet their response time because of insufficient amount of the available resources. We have considered the following events that cause that services do not meet their response time:

1) Required QoS by service-developer is not provided. This can occur because of the insufficient resource availability.
2) Computing node overload. The overload is a condition that a need for resources exceeds devices capabilities, causing undesirable consequences. Factors influencing for overload are:
   a. Background workload (e.g. operating system processes). We assume that the background workload is the constant value on each computer node.
   b. Workload generated by services.

The important remark is that service response time depends on the amount of the input data. Too frequent input data delivery can cause a situation in which service does not meet its response time. Specifying policies according to the input data can prevent this situation. For example, DDS[1] specification describes policy named: TIME_BASED_FILTER. This policy specifies that the service does not receive more frequent data that specified value; regardless of how frequent they are available (DDS_SPEC 04). Figure 1 presents the concept of the computational model.

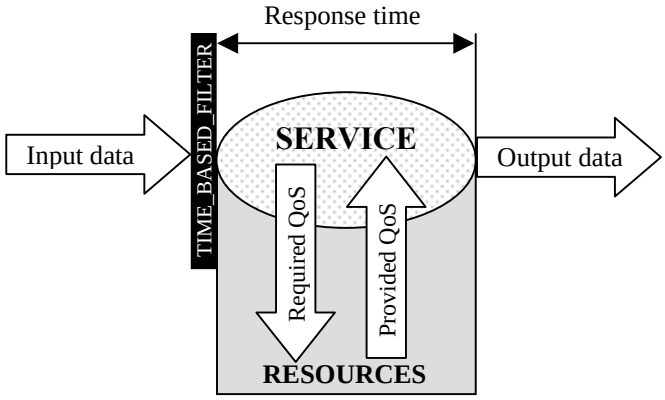

Figure .1 Computational model

In the case where one or more service does not meet their response time, the reconfiguration is performed. The goal of the reconfiguration is to allow a system to evolve incrementally from one state to another at run-time, as opposed to at design-time, while introducing little (or ideally no) impact on the system's execution. In this way, systems do not have to be taken off-line, rebooted or restarted to accommodate changes (Wegdam 03). The reconfiguration needs to transform to the system state, in which all services meet their response time.

The *system state* is defined as the physical services location in a certain moment. It is represented by values of the total exceeded response time, the total overload in the entire system, and total value of all components satisfaction by provided QoS.

The *reconfiguration* is the dynamic reallocation of services (change of physical location). The system state is incrementally transformed to another system state by performing: service, migration or stopping, replacement (Figure 2).

On the presented example, the system state *n* is

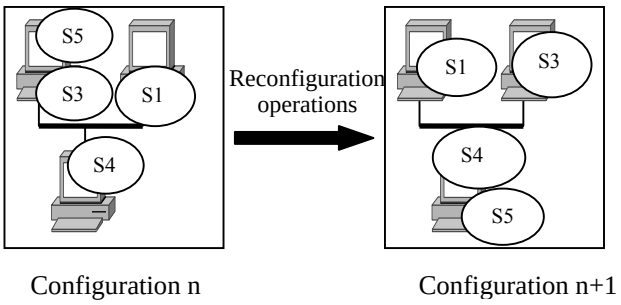

Figure 2. Example of evolving from system configuration n to n+1

transformed to the state *n+1* by performing following reconfiguration steps:

Step1) Replacement service S1 with S3

Step2) Migration of service S5

Problem definition

We consider finding allocation of services, which satisfies all services timing requirements as the dynamic, preemptive scheduling problem. In dynamic scheduling (on-line) the decision is made at run time on the basis of the current request for service. Dynamic schedulers are adaptable to an evolving time task scenario and have to consider only the actual task request and execution time parameters (Mullender93). The scheduler is preemptive because interrupts the execution of the service. The preemption is only done in case of safety assertion that exceeds the scope of this paper.

The dynamic reconfiguration is an intrusive process to the system. The costs of the reconfiguration according to (Santos 01) can be distinguished into:

– Direct costs, which overheads usually CPU time, memory, space and communication bandwidth
– Indirect costs - change of the allocation introduces some delay in the system of reconfiguration process

We state that the allocation algorithm shall propose the reconfiguration that provides services allocation that satisfies all timing requirements, and in minimal degree is intrusive to the system.

Formal Problem Definition

Software services are indexed from 1… *nums*, and nodes are indexed from 1…*numn*. The problem considers allocation of *nums* services to *numn* nodes**.**

[1] OMG Data-Distribution Service (DDS) is an emerging specification for publish/subscribe middleware.

The allocation of the services s_i (where i is the service number, and $i \in 1\ldots nums$) to the node j (where j is the node number, and $j \in 1\ldots numn$) is denoted as pair (s_i,j).

Services allocation is the system state. It is denoted by vector **S**, where $\mathbf{S}=[(s_1,j),(s_2,j),\ldots,(s_{nums},j)]$.

Each service s_i running on the computer node j has resource expenditure.
The function $cost(s_i,j) \in [0,1]$ (where $s_i=j$) returns resource expenditure of service s_i on the node j. For example, let us assume that service (number 1) expenditures of 33 MB memory on the node 2, which has 512 MB of total memory. Then function *cost(1,2)* returns value 0.065 (which is: 33/512).

Each service s_i requires QoS denoted by $qosr_i$.
Each node j provides QoS, denoted by $qosp_j$.
Function $satisfaction(qosr_i, qosp_j) \in [0,1]$ returns value, which expresses degree of services satisfaction by provided QoS. Let us assume that service (number 1) requires 35MB of memory. When the node 2 provides it 30MB, function: *satisfaction(1,2)* would return 0,85 (because of 30/35).

The value *total_satisfaction* is the sum of all services's satisfactions.

$$total\_satisfaction = \sum_i \sum_j \text{satisfaction}(\text{qosr}_i, \text{qosp}_j)$$

Each computing node j is characterized by values:
- Resources capacity[1], defined by value cap_j.
- Background workload denoted α_j

The overload event occurs on the computing node j, when sum of all services costs run on that node, exceeds value of the resource capacity.
We defined the function $overload(j) \in [0,1]$ as follows:

$$\text{overload(j)} = \begin{cases} 1, \ \sum_{i=1}^{nums} \cos t(s_i, j) + \alpha_j > cap_j \ \ if \ \text{s}_i = j \\ \dfrac{\sum_{i=1}^{nums} \cos t(s_i, j) + \alpha_j}{cap_j} \qquad \text{if } \text{s}_i = j \end{cases}$$

The value *total_overload* is the sum of overloads on each node in the system.

$$total\_overload = \sum_{j=1}^{nums} overload(j)$$

The function $idle(j) \in [0,1]$ returns value that expresses idle on the node j. It is defined as follows:

$$idle(j)=1-overload(j);$$

[1] capacity is the maximum resource that can be assigned (allocated) to services.

The value *total_idle* is the sum of nodes low loads.

$$total\_idle = \sum_{j=1}^{nums} idle(j)$$

The function $deadline(s_i)$ returns the value describing how much each service exceeds its response time.
The value *total_deadline* is the sum of values returned by function $deadline(s_i)$ for each service.

$$total\_deadline(S) = \sum_{j=1}^{nums} deadline(s_i)$$

The reconfiguration r (indexed by $1\ldots numr$) is a quadruple $r=(a,s_i,j_s,j_r)$, where
- a – reconfiguration action, e.g., service migration, stopping
- s_i – reconfigured services
- j_{sen} - node, which is the sender of the service
- j_{rec} - node, which is the receiver of the service

The performance of the reconfiguration **r** evolves system state $\mathbf{S_k}$ to system state $\mathbf{S_{k+1}}$.

Vector **R** consists of consequent reconfigurations r: $\mathbf{R_{numr}}=[r_1,r_2,\ldots numr]$
The function $perf\_reconf(\mathbf{S},\mathbf{R_{numr}})$ performs consequent reconfigurations, defined in vector $\mathbf{R_{numr}}$, on the system state **S**. It returns vector $\mathbf{S_{numr}}$ that is the state of the system after $numr$ reconfigurations:

$$\mathbf{S_{numr}} = perf\_reconf\ (\mathbf{S},\mathbf{R_{numr}})$$

The performance of $numr$ consequent reconfigurations has a cost expressed by the function $total\_reconcost\ (\mathbf{S_0}, \mathbf{S_{numr}})$. It returns total cost of performance of reconfigurations defined in vector $\mathbf{R_{numr}}$,

We define the problem as finding the reconfiguration expressed by the vector $\mathbf{R_{num}}$. Performance of this reconfiguration shall:
- Minimize the value of *total_deadline*
- Minimize the value *total_overload*
- Minimize the value returned by function *total_reconcost*
- Maximize the value *total_satisfaction*

Approach

The problem of finding the appropriate reconfiguration may be considered as the optimization problem. The searching optimum is the reconfiguration that fulfils assumed criteria. However some of the optimization criteria may oppose e.g. minimize total exceeded response time and minimize cost of the reconfigurations. Hence, the problem of finding an optimal solution (sub-solution) is stated as the *multiobjective optimization problem*. The goal of multiobjective problem is to find variables, which optimize the objective functions simultaneously; in this

manner the solution is chosen from a so-called Pareto optimal set (Azarm). The optimal solution of the stated problem can be found by searching through all space of possible services allocation. However, in this way stated problem is NP complete. The computational complexity of the search-based problem excludes this approach. Hence, we re-stated the problem as the planning problem. Each plan consists of the consequent steps of the actions (reconfigurations actions). However classical planning algorithms, which use description in some formal language, usually first-order logic or a subset thereof (Russell, Norvig 95) can not be applied to this problem because of the incomplete information. In classical planning, the initial state is completely known, and no information is available from sensors. In planning with incomplete information, the initial state is not known but sensors information may be available at execution time (Bonet and Gener 00). For example, the reason why the service does not meet its response time cannot be fully determined.

Conformant planning, a term coined in (Smith and Weld 98), refers to planning with incomplete information but no sensor feedback. A conformant plan is a sequence of actions that achieves the goal from any initial state compatible with the available information. The problem has been addressed in (Smith and Weld 98) with an algorithm based on the ideas of independent heuristics over a number of problems. From a mathematical point of view, the problem of conformant planning can be seen as the problem of finding a sequence of actions that will map an initial belief state into a target belief state. A belief state in this context is a set of states: the initial belief b_0 is the set of possible initial states, and the target beliefs are the sets that contain goal states only. Actions in this setting map one belief state into another (Bonet and Gener 00).

The problem of planning with incomplete information is no longer a deterministic search problem in belief space. Since the observations cannot be predicted, the effect of actions over belief states becomes non-deterministic, and the selection of actions must be conditional on the observation gathered (Bonet and Gener 00). In order to cope with incomplete information and not fully observability we have introduced the planner based on the belief approach. In this approach certainty (or belief) on a given statement (or event) are represented using probabilistic values, and combine beliefs on a set of statements using probability theory.

Planner

The planner consists of following logical modules:

- Planner generator that creates a collection of plans.
- Planner selector that selects the plan. It evaluates quality of plan. It chooses that one, which in the highest degree transforms, system to the state that satisfies all timing requirements.

The selected plan is applied to the system. The concept of the planner is presented on the figure 3.

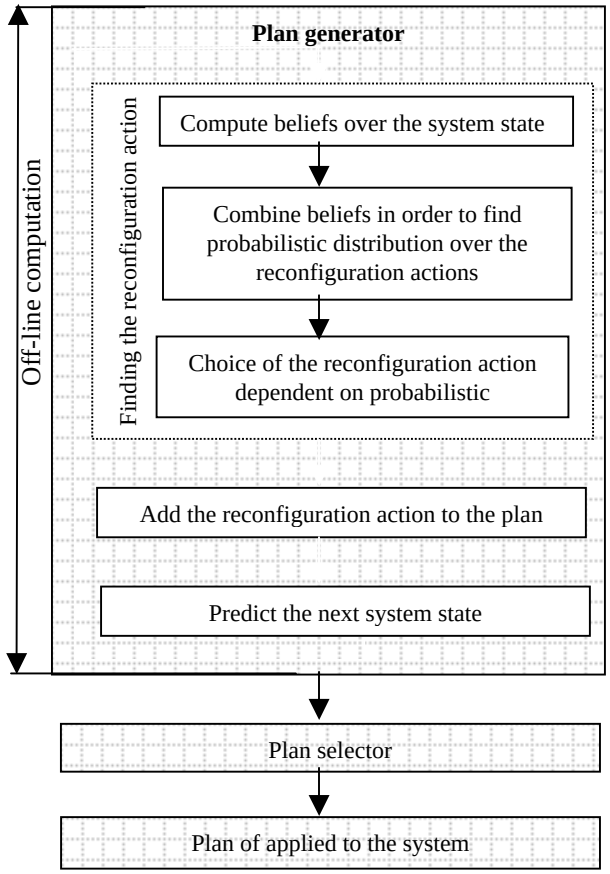

Figure 3. The planner concept.

Plan generator

The plan consists of limited number of reconfiguration actions.

The initial state of the system is the input to the planner. Services location and values describing satisfactions of services by provided QoS are fully observable. Values of resources overload and services responses times come from measurements (incomplete information).

The service *migration*, *replacement*, and *stopping* are operators used by the planner to change the state of the system. The migration is the process of moving the service execution location from one computing node to another. The service replacement is the mutual change of two services execution location. It is realized as the sequence of two simultaneous services migrations.

The plan consists of several reconfiguration actions. These are the output from the planner. The performance of these reconfigurations actions shall bring the system into the state, in which criteria stated in the problem statement will be satisfied. For example, the plan can be expressed as shown in the table 1.

Step #	Operation	Service id #	Node id# (source)	Node id# (dest.)
Step 1	migration	56	34	43
Step 2	migration	23	43	34
Step 3	stopping	5	3	3
Step 4	migration	28	2	7

Table 1 Example plan of the reconfiguration

Each reconfiguration action is a quadruple:

- Operator e.g. migration;
- Reconfiguring service;
- The source node - sender of the service
- The destination node - receiver of the service.

Uncertainty Handling

Data coming from measurements are discounted in the process of information aging. The information aging references to a situation, in which data describing certain state does, not correspond to the current system state, because of data delay collection (Santos 01). Through information aging, data are assessed, in terms of time delay. The assessment is expressed as the belief value of the certain statement. When data are evaluated as a "fresh", high value of belief is assigned to this statement. Obsolete information have low significant to the planner. Hence, low beliefs values are assigned to such a statement. For example, belief of the node overload depends not only on values coming from measurement (e.g., cpu load, free memory), but also from the time of information delivery. The example of information-aging assessment is presented in the figure 4.

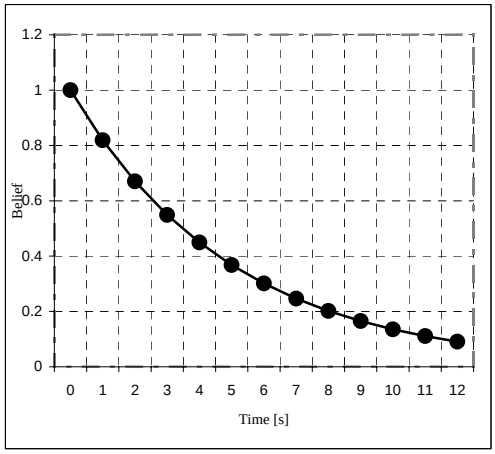

Figure 4. Information aging graph

Exemplary assessment can be derived from following equation:

$$\mathrm{B}_{\text{info-aging}} = \exp(-\frac{time^2}{\tau}) * B$$

$B_{info\text{-}aging}$ - Belief after information aging assessment

B - Belief before information aging assessment

τ - Time constant (in this example 5 sec.)

Beliefs definition

In the process of the information aging the values representing the system state are mapped to the probabilistic values represented by following beliefs:

– Belief of node overload: B_{overj}

– Belief of node idle: B_{idlej}

– Belief of satisfaction of each service on each node: B_{satij}

– Belief that each service exceeds the response time: $B_{deadlinei}$

Belief (B_{ovej}) of overload node j is the fraction of values: overload node $\boldsymbol{j}$ and total overload in the system:

$$\mathrm{B}_{\text{overj}} = \frac{overload(j)}{\text{total\_overload}}$$

Belief (B_{idlej}) of node j idle is the fraction of values: the node idle and total idle in the system:

$$\mathrm{B}_{\text{idlej}} = \frac{idle(j)}{\text{total\_idle}}$$

Belief (B_{satij}) of satisfaction of service i by providing QoS by the node j is the fraction of values: satisfaction of service i and total satisfaction in the system.

$$\mathrm{B}_{\text{satij}} = \frac{\text{satisfaction}(\mathrm{qos}_{\text{ri}}, \mathrm{qos}_{\text{pj}})}{\text{total\_satisfaction}}$$

Belief $(B_{deadlinei})$ of exceeding response time by service i is the fraction of values: value which expresses exceeding response time by service i and value which expresses total exceeding response time in the whole system.

$$\mathrm{B}_{\text{deadlinei}} = \frac{\text{deadline}(\mathrm{s}_{\text{i}})}{\text{total\_deadline}}$$

Each of previously defined beliefs express the degree, how certain value, describing service state, influences the whole system behavior. The key concept is that those services, which in the highest degree negatively influence the system behavior, shall be firstly reconfigured.

Finding the reconfiguration action

We have previously defined the reconfiguration as the quadruple $r=(a,s_i,j_s,j_r)$. At this stage, the planner assigns to the each possible reconfiguration action the belief B_r. This belief expresses the performance suitability of each reconfiguration action for the particular system state. For example, lets assume that $B_{r1} > B_{r2}$. This relation denotes that performance of the reconfiguration r_1 will set the system state closer to the global system goal then performance of the reconfiguration r_2.

Belief B_r is the weighted combination of the previously defined beliefs: B_{overj}, B_{idlej}, B_{satij}, $B_{deadlinei}$. Figure 5 presents how belief B_r is set. This graph illustrates a dependency among the beliefs representing the system state and beliefs, which are calculated in order to find the reconfiguration action.

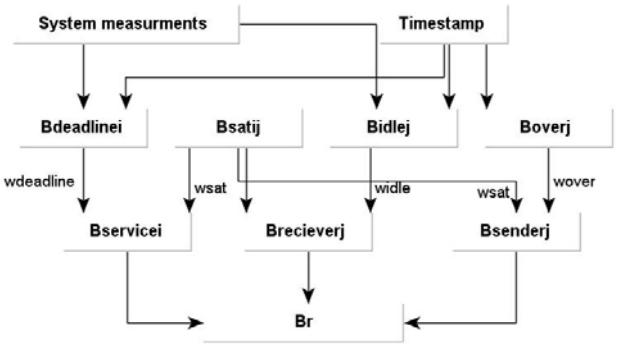

Figure 5. Setting the belief $\mathbf{B}_r$

The belief that node *j* is the sender of the service $(B_{senderj})$ is derived from a weighted combination of two beliefs: belief that node is overloaded (B_{overj}) and belief of the service *i* satisfaction running on node *j:* (B_{satij}).

$$B_{senderj} = w_{over} * B_{overj} + w_{sat} * B_{satij}$$

$$w_{sat}, w_{over} \in (0,1) \quad \text{and} \quad w_{sat} + w_{over} \leq 1$$

w_{sat} – satisfaction belief weight
w_{over} – overload belief weights

Weights values prioritize criteria of the allocation. The priority of the allocation is linearly dependent on those weights.
The belief that node *j* is the receiver of the service $(B_{recieverj})$ is derived from a weighted combination of two beliefs: belief of idle (B_{idlej}) and belief of the service *i* satisfaction running on node *j* (B_{satij}).

$$B_{recieverj} = w_{idle} * B_{idlej} + w_{sat} * B_{satij}$$

$$w_{sat}, w_{idle} \in (0,1) \quad \text{and} \quad w_{sat} + w_{idle} \leq 1$$

w_{idle} – idle belief weight

The belief that the service *i* is suitable for the reconfiguration is derived from weighted combination of two beliefs: belief that the service exceeds $(B_{deadline})$ the response time and belief of service *i* satisfaction (B_{satij}), running on node *j*.

$$B_{servicej} = w_{deadline} * B_{deadlinei} + w_{sati} * B_{satij}$$

$$w_{sati}, w_{deadlinei} \in (0,1) \quad \text{and} \quad w_{sati} + w_{deadlinei} \leq 1$$

w_{sati} – satisfaction belief weight
$w_{dealinei}$ –response time belief weight
Finally, the planner combines derived beliefs of $B_{senderj}$ $B_{servicei}$, $B_{recieverj}$ to assign belief to every possible reconfiguration action.

$$B_r = B_{senderj} + B_{servicei} + B_{recieverj}$$

The example of the probabilistic distribution is presented in the table 2.

Reconfiguration #	Operation	Service id#	Node id# (source)	Node id# (dest.)	$\mathbf{B}_r$
1	stopping	16	12	12	0.16
2	migration	32	43	34	0.25
3	migration	35	4	17	0.32
4	stopping	18	12	12	0.12
5	migration	3	27	9	0.15

Table 2 Example of the probabilistic distribution over reconfiguration actions

The final step of finding the reconfiguration action is the selection the action from the collection of all actions. This selection is done based on the value of belief B_r.
Several selection strategies are possible. The most simple and intuitive strategy is the selection of action with a maximal value of belief B_r. However, we received more satisfying results using a strategy, which randomly selects action with likelihood linearly dependent on the value of belief B_r. The selected action is added to the plan as the following step.
The plan generator updates beliefs representing the system state, based on the prediction how the selected action may influence the system state. This prediction of the new system state is non-deterministic and dependent on the heuristics of the computational model. Hence, we cannot determine any generic model of new system state predictor. The following, simple example shows how the plan generator predicts the new system state in our computational model. Let us consider that selected reconfiguration action is the service 4 migration from node 2 to 3. Then planner updates beliefs over the system state. It decreases load on the node 2 by load generated by service 4 and adds it to the node 3.
After updates of beliefs values representing system state, planner searches for the next reconfiguration action for the new anticipated system state.

Plan selector

The presented reasoning of finding the reconfiguration actions is not a deterministic process. Each generated plan cannot be necessarily the solution of the problem (the new allocation of the services may not schedule the optimal response time). Different plans (consisting of different reconfiguration actions) are generated based on the same

values of beliefs representing the initial system state. The amount of generated plans for each initial state is the input parameter to the plan selector. The module of the *plan selector* selects the plan (solution), among many generated plans, which transforms an initial belief state into a target belief state. The selection of the plan is performed, based on the following criteria of plan assessment:

- Minimal (total_deadline) – minimal total value of the exceeded response time.
- Minimal (total_overload) - minimal total value of the overload in the system.
- Maximal (total_satisfaction) – maximal value of components satisfaction by provided QoS.
- Minimal (total_reconcost) - minimal value of the reconfiguration cost.

Simulation

We have developed the simulation program in order to demonstrate the concept of the service allocation planner. It simulates the behavior of the reconfiguration service for the publish/subscribe middleware (Zieba 05 et al). The service was designed in the "manager-agent" pattern architecture. The agent monitors load on the nodes and controls the services. The manager is the single, centralized program, which takes decision about the allocation of the services. The allocation algorithm is implemented as the presented planner.

The simulator consists of the following parts:

- An emulator of the reconfiguration system. The system consists of nodes, with syntactical generated workloads, and services operating on these nodes.
- The reconfiguration service, that changes allocation of the services
- Service allocation planner, as presented in this paper.

Results

The simulated scenario demonstrates the concept of the service allocation planner. It consists of 10 nodes and 50 services. One of the nodes has a failure. The reconfiguration system re-instantiates 7 services operating on this failure nodes on the rest of available nodes.

We set the plan generator to maximal 20 reconfiguration actions and maximal 20 generated plans.

The figure 6 presents how performance of number of reconfiguration actions influences the system state. Each of dots represents the state of the system after performance of reconfiguration action. In the initial system state response time of all services is exceeded in 0,05 sec. The value of the overload is normalized (between 0-1), and at the beginning is 1. After 19th of reconfiguration action the exceeded response time and the total system overload are zero.

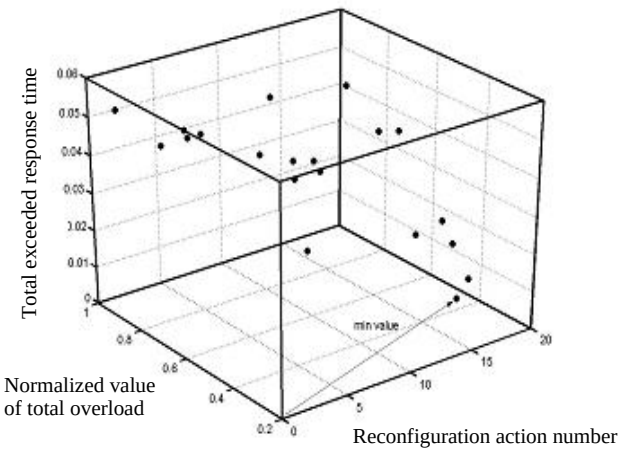

Figure 6. Performance of the reconfigurations actions in the plan

The plan selector compares plans in order to select the best plan (solution), see figure 7 for details. Each dots denotes the output of each plan. Only plans 4th, 2nd, and 9th completely eliminate values of exceeded response time and total overload. Plan 9th is achieved after performance of 19 reconfiguration actions, 4th after 17 actions, and 2nd after 16 actions. In this case, we define the reconfiguration cost as the amount of the performed reconfigurations actions. All of these mentioned before plans achieve goal of the reconfiguration, but selector chooses the 2nd because the reconfiguration cost is the lowest.

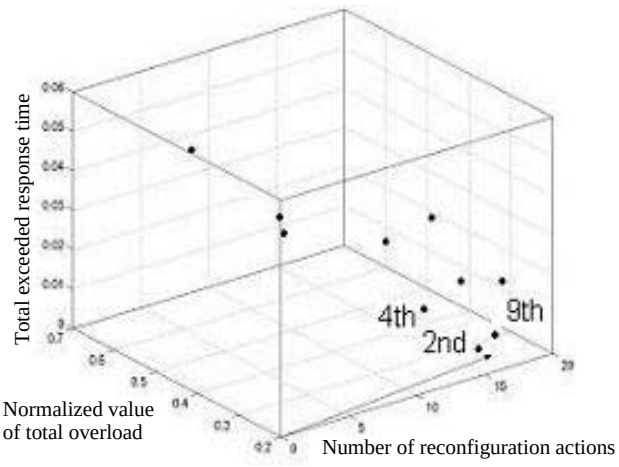

Figure 7. Comparison of the generated plans

Conclusion

In this paper, we presented the services allocation planner for the dynamic reconfiguration. The classical planning algorithms cannot be applied to this problem because of the incomplete information. We deal with this issue using the conformant planning. The finally selected plan consists of a sequence of reconfigurations actions that achieves the goal from any initial state compatible with the available information. The goal of the planner is finding the reconfiguration that eliminates exceeded services response time. Satisfying all timing requirements is the dynamic, preemptive scheduling problem. The reasoning about finding the most appropriate reconfiguration actions and plan evaluation was explained. The simulation results illustrate the concept of the planner.

Acknowledgements

This work was supported by European Research Program: Marie Curie Host Fellowship under the contract number: HPMI-CT-2002-00221

References

Azarm S.- Multiobjective Optimum Design, website: http://www.glue.umd.edu/~azarm/optimum_notes/multi/multi.html

Bonet B., Gener H. 2000- Planning with Incomplete Information as Heuristic Search in Belief Space

DDS_SPEC 2004- Data Distribution Service for Real-Time Systems Specification, ptc/03-07- 07

Engelmore, R., and Morgan, A. eds. 1986. Blackboard Systems. Reading, Mass.: Addison-Wesley.

Fluckiger, F., 1995 - Understanding Networked Multimedia(Ed, ITU) Prentice Hall, , pp. 338.

Muhl G. - Large-Scale Content-Based Publish/Subscribe Systems – Dissertation Vom Fachbereich Informatik der Technischen Universitat Darmstadt 2002;

Mullender S. 93 – Distributed Systems, ACM Press New York
Bonet B., H. Geffner, 2000 - Planning with Incomplete Inforamtion as Heurisitc Search in Belief Space.

Rechtin E., Maier M. W. 97 - ,The art. of systems architecting 1997 by CRC Press, Inc, ISBN: 0-8493-7836-2

Russell S.J, Norvig P. – Artificial Intelligence – A Modern Approach – Prentice-Hall International, Inc, 1995

Santos L.P.P 01- Application Level Tun Time Load Management A Bayesian Approach – PhD Thesis, Universidade Do Minho 2001

Smith, D., Weld, D. 1998. Conformant graph plan. In Proceedings AAAI-98, 889{896. AAAI Press.

Wegdam M. 03 – Dynamic Reconfiguration and Load Distribution in Service Middleware – PhD thesis, University of Twente, CTIT Ph.D-thesis series, No. 03-50, ISSN 1381-3617; No. 03-5, 26 June 2003.

Wolfy A.L., Heimbignery D, Knightz J., Devanbu P., Gertz M., Carzaniga A. - Bend, Don't Break: Using Reconfiguration to Achieve Survivability.

Zieba B., Glandrup M., Sinderen M., Wegdam M. 05- Reconfiguration Service for Publish/Subscribe Middleware Systems - Submitted for Euro-Par 2005