Self-management as Support to an Advanced Traveler Information System

Marcia Pasin[1], Felipe Silvano Perini[1], Ana L. C. Bazzan[2]

[1]Centro de Tecnologia – Universidade Federal de Santa Maria (UFSM)
Av. Roraima 1.000 – Cidade Universitária – Santa Maria – RS – Brazil

[2]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{marcia, fsperini}@inf.ufsm.br, bazzan@inf.ufrgs.br

Abstract. *Advanced traveler information systems (ATIS) are growing in popularity posing new challenges to distributed infrastructures. Services and data tend to be distributed, heterogeneous, and there are issues related to elastic and increasing demand, failures and load peaks. Management of such infrastructures is a complex task. To address these issues, a promising approach is to implement system management as an autonomic service. In this paper we propose a self-managed system (SMS) to support a distributed infrastructure running an ATIS. Based on current and previous status, the SMS adapts itself to achieve efficient service and high availability. Experimental evaluation showed that the system is efficiently reactive due to changing demand and occurrence of faults.*

1. Introduction

Well established processes in traffic and transportation are being transformed by the rapid growing availability of data, as well as by the ever increasing demand for services, especially those related to mobile devices. One example is the demand for advanced traveler information systems (ATIS), one important part of intelligent transportation systems. Although ATIS has been conceived primarily for private transportation (e.g. route guidance, hazards broadcast, en-route planning, etc.), it has been increasingly changed to accommodate transit users as well. For instance, users of mobile devices may now access information about timetables for buses, trams, etc., which serve a given location. Given the ID of a bus stop or the user's GPS location, this and other services can be provided by an information system (IS). This solution is interesting because it operates over a popular device and it does not require expensive investments in infrastructure.

One issue here is that such an IS is subject to load peaks, with thousands of users requesting information at the same time, while at some periods the demand drops drastically. Also, currently, system management is in charge of human beings, thus leading to configuration errors and low availability. To address these issues, a promising approach is to implement system management as an autonomic service [Kephart and Chess 2003]. This can be used to manage a group of servers over a distributed infrastructure running the IS/ATIS. Advantages of this approach include no human intervention and adaptation despite failures, load peaks and high number of connected clients.

In this paper we propose a self-managed service (SMS) to support a distributed infrastructure running IS/ATIS. Our SMS is based on a distributed monitoring sys-

tem that collects information from servers in the infrastructure and triggers a reconfiguration mechanism whenever necessary. Roughly, the SMS adapts the system upon load peaks and failures. In contrast with other implementations [Bouchenak et al 2006, Saleem and Chen 2008], our SMS has a fully distributed implementation. Indeed, the SMS is composed by a several independent self-services. Therefore, a conflict resolution mechanism is required to allow progress despite of antagonist behavior of independent services. This work is being supported by the Brazilian Government agency FAPERGS/CNPq (Project RS-SOC - Rede Estadual de Simulação Social) under grant number 10/0049-7.

2. Context

An ATIS that is dedicated to mass transit processes information about buses and other rapid transit systems, as well as routes for users and eventually also for mass transit providers using information stored on a database. For sake of illustration consider the following scenario. A bus leaves the garage and follows a previously established route/timetable. The route is composed by a set of bus stops. Bus and bus stop are uniquely identified using IDs. Each time a bus reaches a bus stop, it sends its ID, the bus stop ID, and the time in which the bus stop was reached to an IS. The IS calculates the time the bus will reach the next bus stop. Users (travelers) can then access this information using their PC at home or portable devices such as mobile phones, smart phones, and PDAs. Supporting this ATIS, there is a hardware infrastructure organized as follows: (i) a set of sensors to detects buses events, (ii) a group of computers (nodes) to execute replicated servers instances running the IS/ATIS, (iii) a load balancer to distribute requests among severs following round-robin policy and (iv) mobile phones, smart phones, and browsers widely spread.

Bus data (such as current position and speed) are collected by GPS devices located in each bus. A micro-controller installed in each bus collects buses' data and sends them to the IS via mobile phone technology (GSM/GPRS) in a real-time fashion. When the message arrives the server, the data are used to update database with current information about buses' positions and other issues.

The IS/ATIS has as features: (i) servers in the infrastructure are stateless, (ii) databases associated with servers hold one table per bus line, (iii) users interact with the infrastructure via request/reply protocol implemented by a read-only transaction, and (iv) bus sensors' interact with the infrastructure via write-only transactions.

The IS was partially implemented in a previously work [Bastos and Jaques 2010], and is currently being extended to allow all features here described. When it becomes operational, one issue that will probably arise is that such an IS is subject to load peaks, with thousands of users requesting information at the same time, while at some periods the demand drops drastically.

3. Self-managed system

On the top of the hardware infrastructure previously described, the SMS manages application servers using information collected in the infrastructure. Our SMS is based on event correlation: events are detected in each server and are analyzed using a set of pre-defined rules. Eventually these events fire rules that cause the application of actions. Currently,

analyzers are decentralized to avoid single point of failure. So, conflicts may occur if a set of events matches a set of two or more rules in different analyzers. A strategy to detect and solve conflict is required. In a meta-level, agents are used to decide about conflicting situations.

3.1. Architecture

Our SMS adapts a set of distributed application servers. In each server, we have the following components: (i) sensors which collect information about new events (e.g. information about a failure in a node, a load peak, etc.), (ii) agents which encapsulate analyzers and communicate each other through messages to resolve conflicting actions, (iii) analyzers which are designed to match information provided by sensors and rules stored in databases, and (vi) actuators which are responsible for applying the actions (i.e. calling services) triggered by the rules such as activating a new node and removing an existing one. Note that sensors and events at the SMS level are not the same sensors/events used by the application level. Figure 1 shows the components of the SMS in the middle of the agent-based system and application layer.

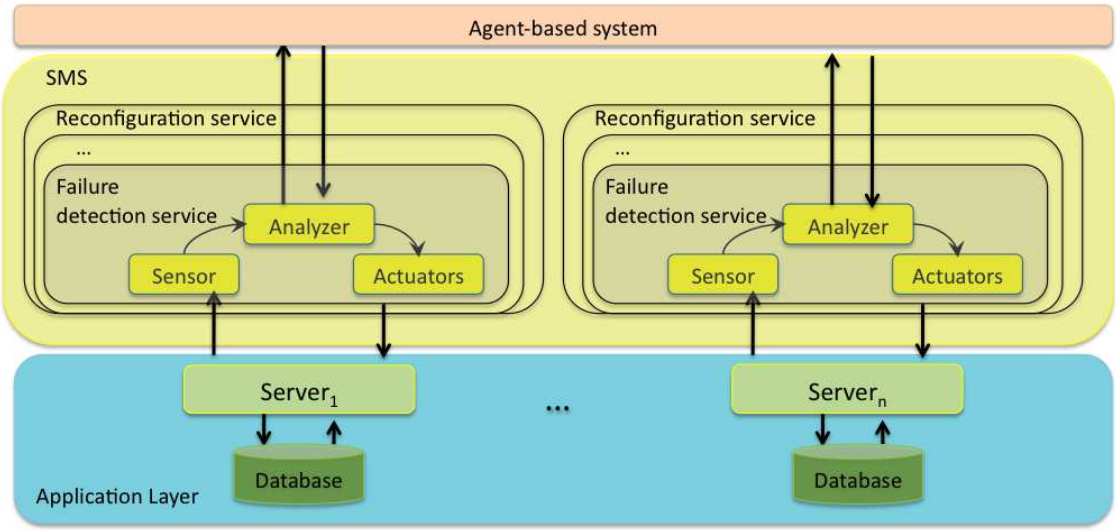

Figure 1. The Self-managed system as middleware service

3.2. Self-managed services

The SMS comprises a set of services to implement self-optimization and self-healing properties. Roughly, self-optimization means changing the number of current active servers in face of elastic demand. This property is mainly implemented by the reconfiguration service. Self-healing means restoring the service in presence of failures. This property is mainly implemented by the failure detection service. If a server crashes, it will be unavailable to serve users requests.

Apart from reconfiguration and failure detection, other services are used to manage the group of servers such as load balancing and state transfer. Each service is executed by an independent module and is dispatched by a different actuator. In the following, we give an overview of these services which are summarized on Table 1. The current implementation of some services (marked with a "*" in Table 1) is attached to the application-

context to allow some level of quality of service. Other implementations are generic and can be easly reused in other contexts.

Table 1. Self-management services

Service	Description
Failure detection	Detection of crashed nodes
Load peak and idleness detection	Detection and prediction of node load peaks and idleness
Node reconfiguration	Removal and addition of an application server
State consistent replication*	Ensuring database consistency
State transfer*	Updating the state of a new application server recently added in the group of servers

Failure detection service. This service detects faulty servers in the IS, using a heartbeat mechanism. Each server in the infrastructure continuously advertises to the failure detection service that it is alive, every predefined period of time. When a heartbeat is missed, the node is declared as failed by the failure detection service. Consequently, this server is marked as crashed in the list of operational servers and it does not receive clients requests any more. The current implementation of this service uses the support of a group communication system.

Load peak and idleness detection service. This service is implemented by a local and a global module using a double time series scheme. Locally, each server in the group is periodically monitored (evaluating the CPU load). Thresholds are previously defined to limit maximum and minimum values to CPU occupation in nodes. Taking into account these values, if the low threshold is achieved, the node is marked locally as idle. If the high threshold is achieved, the node is marked as overloaded. These values are taking in a local time series vector. The global module takes into account values collected locally in all servers, using the group communication system support, as well previously stored information using a global time series vector. The scheme of double-level of time series is used to avoid the addition or removal of a node due to a momentary high/low load peak.

Reconfiguration service. This service starts and stops servers in the server group and acts with the support of both failure detection and load peak and idleness detection services. A new application server is started in an available node in the infrastructure (if there is a node available) in face of high load peak or failure detection or stopped if the system is idle. Nodes, which are not executing an instance of an application server, can be reallocated to other purposes. The number of active servers is always less or equal of the number of operational nodes in the infrastructure. If this threshold is reached, the system starts to deny client's request.

State consistent replication service. Taking into account application features reported on section 2, the current implemented database consistency approach is lazy. A blind write is multicasted to the group of servers and is individually executed in each database each time a new sensor information arrives in the infrastructure. So, each server updates its table using no consistency approach. Therefore, clients would possibly get old information (i.e. 5 min to get a bus instead of 4 min). However, we do not consider this a problem since sensor information arrives frequently in the infrastructure and the information sent to passengers will be always an approximation of the actual bus position.

State transfer service. An application server that was out of service for a time

period (due failure or idleness) should recover its state using a procedure called state transfer. During the execution of this approach, the server gradually requests the current version of data items stored in other database server. Aiming not overcharging the client service with additional communication, and due to the characteristics of the ATIS, the recovered server can request updating only the last data items (i.e. only the information concerning the last bus which achieves a suitable bus stop).

4. Implementation and evaluation

The SMS is being implemented using Java program language and JGroups as group communication system support. Currently, great part of the self-managed services were implemented, except state transfer and state replication services. More details about the implemented services can be found in [Pasin et al 2011].

To evaluate our implementation, we defined a set of tests executed under different scenarios. Those tests were executed in a set of three ordinary computers running ubuntu/linux platform connected under 100Mbps Fast Ethernet. Due to space limitations, were we report only our initial test facing the reconfiguration service with a growing load peak. Figure 2 shows graphics obtained in a scenario of tests with three nodes and a variable number of clients requests (from 100 to 1000).

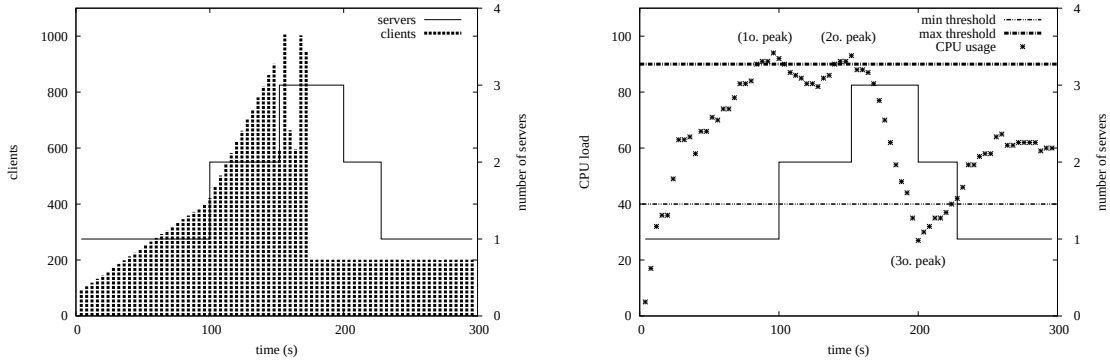

Figure 2. Reconfiguration of the infrastructure due a change in the system load

Initially, there is only a server executing in a single machine which is processing client requests. So, the server is charged with more and more client requests until the CPU load reaches a threshold (90%). The load is increased to overload the initial configuration. So, automatic initialization of a new server occurs at another node, if the CPU rate reaches 90%. The startup of new servers is repeated until the maximum number of available nodes in infrastructure is achieved. Similarly, if the load decreases (in the experiment it was reduced and maintained at 200 active clients), the CPU occupation rate also reduces and when it reaches 40%, servers are deallocated.

5. Future works

The total system implementation is still not finished. Future works include improving the module of detection of load peaks and idleness with a predictive approach (such as [Sant'Ana et al 2010]). In the current implementation, there is a two-level time series vector taking into account only the CPU occupation in each sever but other metrics such

as memory occupation, and number of current requests' in execution can be used in future implementations.

Finally, self-managed services previously reported can be conflicting. For example, two or more decisions taking by concurrent services could lead to inconsistent actions. Conflict resolution is a challenge for self-managed systems. In the current implementation, if agents perceive that services are proposing inconsistent actions, they vote to reach agreement. Therefore, our aim is not only to use already implemented techniques but also to explore more sophisticated agent-based techniques such as negotiation. Negotiation among agents can be implemented in a variety of ways (auctions, bargaining, arguing, etc.). Also, negotiation opens up the possibility of including a reputation-based mechanism where agents assign a reputation degree to each other, thus affecting the voting protocol.

6. Conclusions

ATIS have gained wide-spread popularity. This kind of service typically is distributed or replicated on clusters to allow availability and scalability. Managing such infrastructure is complex and costly since services are subject to elastic demand, failures and load peaks. Autonomic computing is believed to alleviate the complexity of the management on these systems maintaining a suitable service with regard to system changes. In this paper, we have proposed a SMS able to deal with these issues. Its main and novel features are: providing ATIS-related services; accounting for time series to avoid unnecessary reconfigurations; and conflict resolution by means of negotiation. Future works include improving the module of detection of load peaks and idleness with a predictive approach and use of reputation tied to the voting process to improve the task of conflict resolution.

References

Bastos, R. and Jaques, P. (2010) ANTARES: Um Sistema Web de Consulta de Rotas de Ônibus como Serviço Público. Revista Bras. de Computação Aplicada, v.2, pp. 41–56.

Bouchenak, S., De Palma, N., Hagimont, D. and Taton, C. (2006) Autonomic Management of Clustered Applications, In: Proc. Cluster 2006, Barcelona, Spain. Sept. pp. 1–11.

Kephart, J. O. e Chess, D. M. (2003) The Vision of Autonomic Computing, In: IEEE Computer, vol. 36. pp. 41–50.

Pasin, M., Perini, F. S. and Bazzan, A. L. C. (2011) Towards a Self-managed Distributed Infrastructure to an Advanced Traveler Information System. In: Anais do Autosoft / CBSoft 2011. São Paulo - Brazil. pp. 33–39.

Sallem, S. K. and Chen, S.-C. (2008) Towards a Self-configurable Weather Research and Forecasting System, In: Proc. ICAC 2008, pp. 195–196.

Sant'Ana, C. H., Leite, J. C. B. and Mossé, D. (2010) Previsão de Carga para Economia de Energia em Aglomerados de Servidores Web. In: Anais do SBRC 2010. Gramado - RS, Brazil. pp. 683–696.