2011 Third IEEE International Conference on Coud Computing Technology and Science

Goal-oriented Self-management of In-memory Distributed Data Grid Platforms

Liliana Rosa
INESC-ID, Instituto Superior Técnico,
Universidade Técnica de Lisboa
Email: lrosa@gsd.inesc-id.pt

Luís Rodrigues
INESC-ID, Instituto Superior Técnico,
Universidade Técnica de Lisboa
Email: ler@ist.utl.pt

Antónia Lopes
LASIGE, Faculdade de Ciências,
Universidade de Lisboa
Email: mal@di.fc.ul.pt

***Abstract*—This paper addresses the self-management of in-memory distributed data grid platforms. A growing number of applications rely in these platforms to speed up access to large sets of data. However, they are complex to manage due to the diversity of configuration and load profiles. The proposed approach employs an adaptation policy expressed in terms of high-level goals to facilitate the task of the system manager, and address the complexity issues posed by the management of multiple configurations. The approach is validated experimentally using the open-source RedHat´s Infinispan platform.**

I. Introduction

Today, many services such as Twitter, Slashdot, Facebook, and Wikipedia, among many others, rely on in-memory distributed data grids to substantially speed up their websites. Distributed data grids supply applications with a scalable storage repository where data can be accessed without bottlenecks and shared across a pool of virtual servers. Platforms such as Memcached [1], Infinispan [2], Coherence [3], Scale Out [4], or Velocity [5], provide fast access to data, decoupling the management of persistence from the critical request processing path. By dramatically improving the deployment of scalable applications, distributed data grids play a key role in cloud-based infrastructures.

In-memory distributed data grids, or simply IMDDGs, are able to adapt their operation in response to changes in the workload. In fact, on-demand elasticity is one of the main features of any middleware platform for cloud computing. These platforms offer several configuration options that can be adapted with significative impact on the system performance. The scaling out process is an adaptation that allows to respond to sudden changes in demand, for instance, flash crowds. This adaptation adds or removes resources as necessary, providing the elasticity so necessary in cloud computing. The elasticity can be obtained using many aspects such as cache size, number of cache cluster nodes, number of copies of each object, underlying communication protocols, eviction algorithms, the locking and deadlock detection schemes, just to name a few.

We propose an approach that allows to autonomously manage the many configurable settings of the platform and other associated middleware. The approach relies in high-level goal policies to guide the self-management of the platform settings. The policy is defined by a system manager, that is aware of the application needs, while the configurations are provided by the platform developers, experienced with its behavior. In this paper, we focus on aspects related with data replication, as we consider that fault-tolerance is also a fundamental aspect to manage in cloud-computing platforms.

The contribution of this work is an approach for the self-management of IMDDGs which provides a clear separation of system manager and platform developers functions. This separation allows to offer the adequate abstraction level for each, while providing an effective manner to tackle the complexity of self-management. This contribution is validated using a prototype of the approach on top of Infinispan (by RedHat).

II. IMDDGs

IMDDG platforms are rich in configurable settings, resulting in a wide variety of system behaviors, that influence significantly the system performance. Thus, the management of so many behaviors becomes overwhelming when the system is subject to variable load. In this section, we address the metrics and the reconfiguration mechanisms more relevant for the aspects related with data replication. The key performance indicators (KPIs) for IMDDGs used in our experimental evaluation are the following.

- *Service Ratio (SR)*: Let *RI* be the rate of incoming (read and write) requests from the application to the IMDDG, i.e., the current load on the system. Let *RS* the rate at which these requests are served, i.e., the throughput. The service ratio is defined as $SR = RS/RI$. Ideally, the service ratio should be 1.
- *Resource Consumption (C)*: This metric captures how many resources are used to maintain the cache operational. As noted before, it is important that the IMDDG is able to scale elastically with demand to save resources.

In terms of reconfiguration, different implementations of IMDDGs offer different reconfiguration mechanisms. The adaptation of the *number of servers/replicas* is cross-platform. More servers may support additional load (increasing the service ratio, if below 1) but they consume more resources. Thus, it is of interest to only have the strictly needed servers.

III. Self-management of IMDDGs

The aim of this approach is to automate the management of the configurable settings of IMDDGs, where adaptations are selected and deployed automatically in response to runtime

978-0-7695-4622-3/11 $26.00 © 2011 IEEE
DOI 10.1109/CloudCom.2011.90

changes. This avoids the complex task of manually determining which system configuration is the more appropriated for the current execution conditions. The approach's planning phase targets the complex trade-offs identified previously, making a clear distinction between guiding the system management and achieving it. This distinction allows the manager to focus on the high-level management of the system and benefits from the expertise of developers in terms of platform configuration and its impact on performance.

A. Architecture

The approach relies on an external controller that follows a control loop model [6], [7], where the main activities performed by the control layer are *i)* the collection of relevant data from sensors; *ii)* the analysis of the collected data; *iii)* the decision on how to adapt the system to reach a desirable state; and *iv)* the implementation of the decision via the available effectors. The main activities of the control loop are carried out by three components: *Monitor*, *Planner*, and *Executor*. The *Monitor* is independent of the IMDDG platform and collects data captured by several *sensors* present in each node of the cache cluster. The *Monitor* also analyzes this data and detects when the system is in an undesirable state. When such a state is identified, the *Monitor* notifies the *Planner*, triggering an event carrying the relevant state information. In reaction to such notifications, the *Planner* determines how to adapt the system and, once a decision is made, it passes this decision to the *Executor*. The *Executor* controls the adaptation process, relying in a number of *effectors*, which exist at every node and implement the reconfiguration mechanisms described in the previous section.

B. Planning based on Goal Policies

The proposed approach employs high-level goal policies [8], [9] to define the behavior goals for the system. This choice offers several advantages. One is that it allows to express the management guidelines independently of the used platform. Additionally, the goals can be changed without affecting the remainder of the controller. Another advantage is that it allows to explore the many trade-offs of the reconfigurable options in terms of performance. Also, it allows to express behavior goals not related with performance; this is useful to express the self-* properties of the autonomic system, such as the self-healing property, by maintaining a level of redundancy to survive failures and address failovers. Finally, this approach is flexible enough to change the management guidelines without changing the adaptations or the effectors.

While there are other approaches that also employ goal policies to achieve an autonomic behavior [8], [10], they do not address distributed systems. IMDDGs not only are distributed but their adaptation depends on the system topology, making the distribution an unavoidable aspect. This is also the case of [9], which led us to evolve the approach to support IMDDGs. Among other features, the new model introduces *scopes* for monitoring and actuation in a distributed setting, with system, component, node, and instance-level scopes. The set of adaptations was also expanded to consider new adaptations and targets. Finally, the adaptation selection process was revised, in order to take into account the new model, adaptations, and other aspects associated with the distribution.

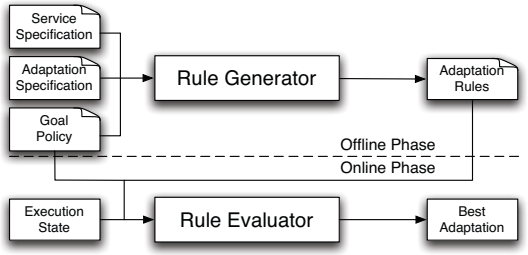


Figure 1. Overview of goal based planning

Figure 1 presents an overview of the planning phase based on goal policies. The planning encompasses an offline and an online phase. During the *offline phase*, a set of adaptation rules is generated from a goal policy and a specification of the available adaptations. Each rule defines a collection of adaptations that might correct a particular deviation in the system behavior. The rule generation process itself is executed in two steps. The first step determines the types of deviations that may affect the system behavior. The second step determines the adaptations that might help to correct each deviation. During runtime, when a deviation occurs, an event is triggered and forces the execution of the *online phase*. The rule triggered by that event is evaluated against the current system state and the policy. One or more adaptations are selected as a result. Next, we describe in more detail the key elements of the planning phase.

1) KPIs: The KPIs allow to characterize the system behavior and assess its state. The specification of a KPI includes its name, the value type, a function defining how the global value is calculated from local values, and the acceptable error margin (*Error*) in any evaluation of the KPI (any two values are equivalent if the distance is below the margin). Four types of KPIs can be defined: *system*, *node*, *component* and *instance-sensed*. The values of *system-sensed* KPIs are measured for the entire system as a whole. The values of *node-sensed* KPIs are measured by individual node and its specification includes an *aggregation function AF*, which defines how the value of the KPI for the entire system is obtained. The values of *component-sensed* KPIs are measured by individual component as a whole and its specification includes a *combination function CF*, which defines how the KPI value for the entire system is obtained. Finally, the values of *instance-sensed* KPIs are measured per component instance, i.e., by individual component in each node. The specification includes a *combination function* and an *aggregation function*. Some examples of KPIs can be found in Section IV. The approach also allows the definition of *composite KPIs* (CKPIs), whose values are calculated using a *Join Function* of other KPIs.

2) Components and Adaptations: The component specification includes the description of all the components available for use in the system. This description includes any con-

Type	Actions	
Component	c.setParameter(param,value)	c.replaceBy(c')
Instance	n.c.setParameter(param,value)	n.c.replaceBy(c')
Node	n.addComponent(c)	n.removeComponent(c)

Table I
ADAPTATION ACTIONS PER TYPE OF ADAPTATION

figurable parameters. The adaptation specification describes the adaptations that can be used to manage the system. Adaptations are defined in terms of a fixed set of actions. To address both distributed and non-distributed components, three groups of adaptations were considered, characterized by their scope: *component*, *instance*, and *node* adaptations. Component adaptations target a component c, affecting all the instances in the system. Instance adaptations only affect an instance of the component c in a particular node n. Finally, node adaptations affect only a node n. The accepted actions per adaptation type are depicted in Table I. Each adaptation also declares the impact of the declared actions on the system KPIs and CKPIs and the time for stabilization. The impact is an estimate of the values of affected KPIs after the adaptation. Examples of adaptations are provided in Section IV.

3) Goal Policy: Goals are the high-level directives that will guide the system management. These goals describe what is the acceptable behavior in terms of KPIs values. There are three types of *exact goals* and three types of *approximation goals*. The *exact goals* separate the values of a KPI in two disjoint sets: *acceptable* and *not acceptable*. An *above* goal will only find acceptable the values above the threshold. A *below* goal will only accept the values below, and a *between* goal only the values in the specified interval. The *approximation goals* are best effort goals that specify a total order between the values of a KPI. A *maximize* goal states that the largest is the best, while a *minimize* goal aims at the smallest. A *close* goal tries to keep the value as close as possible to the *aim* value. The system will try to optimize its behavior with respect to approximation goals periodically (every *period* of time) but there is a *minimum gain* for an adaptation be worthwhile. The six types of goals follow:

```
Goal goalName:kpiName Above threshold_down
Goal goalName:kpiName Below threshold_up
Goal goalName:kpiName Between threshold_down threshold_up
Goal goalName:kpiName Close aim MinGain value Every period
Goal goalName:Minimize kpiName MinGain value Every period
Goal goalName:Maximize kpiName MinGain value Every period
```

4) Rule Generation and Evaluation: The rule generation process consists in two steps that aim to generate the rules that will be evaluated to select adaptations. A rule is composed by an event and a set of combinations of adaptations. Its generation starts by extracting the relevant events from the goals. Next, for each event, the set of possible useful adaptations is selected, i.e., those that help correcting the problem signaled by the event. For instance, assuming that we have an event that demands to lower the KPI (because it is above the limit), all the adaptations that decrease the KPI are selected. The adaptations whose helpfulness is unknown are also selected. Then, all combinations of adaptations that can be performed simultaneously are calculated.

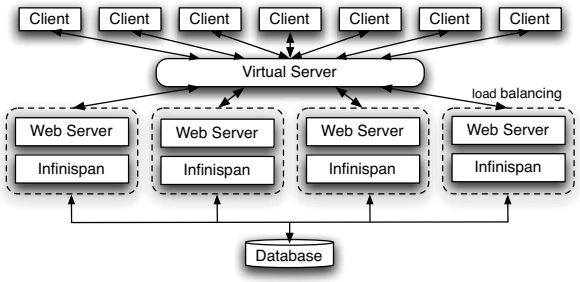


Figure 2. Case study architecture

The rule evaluation process analyzes one rule at a time, using the system state as input. It starts by estimating the impact of the adaptations in the KPIs. Afterwards, the selection algorithm evaluates the different combinations of adaptations against the goal with the highest rank. If a combination fulfills the goal, it passes to the next step, otherwise, it is discarded. If none of the combinations fulfills the goal, they all pass to the next step, as the KPIs associated with lower ranked goals may still be corrected or improved. After, the combinations that passed are evaluated against the next goal, and so forth until all the goals are evaluated. Among the combinations that reach the last step, one is selected to be performed.

IV. CASE STUDY

The case study developed to validate the proposed approach relies in a website, that serves static and dynamic content. The website is served by several web servers, each running on its own node. The website load is balanced by the several web servers. The system relies in an IMDDG to speed up web servers' access to data, with a local instance serving each of system web servers. The local instance has its own cache and serves as a proxy to query and update remote caches of the platform. The website content is generated from data that is available in the cache of the IMDDG platform. If the data is not available in cache (local or remote), it will be fetched from the database and added to cache. In addition to the web server and the IMDDG instance, each node also includes a communication toolkit that provides communication and coordination support among IMDDG instances. In our prototype we use Infinispan [2] as an IMDDG platform. Figure 2 illustrates the case study architecture. The management of the database is not considered in this case study.

The case study's self-management aims at taking advantage of the reconfigurability of the IMDDG platform. The goal is to optimize performance in order to maintain user satisfaction, but at the same time to minimize resource consumption to cut on costs. Furthermore, we are also concerned with healing properties. The development and evaluation of the case study will focus on these goals.

A. KPIs and Goal Policy

The following objectives guide the self-management. One is to self-optimize performance, another objective is to be energy-efficient, thus, cutting on costs. Finally, the last objective is fault-tolerance, providing self-healing and self-

Type	KPI	Values	CF	AF	Error	Description
System	#servers	int	-	-	0	servers
Instance	RI	double	Sum	Sum	0.2	load
Instance	RS	double	Sum	Sum	0.2	throughput

CKPI	Values	JoinFunction	Error	Description
service_ratio	[0,1]	RS/RI	0.01	service ratio

Table II
THE KPIS AND CKPI OF THE CASE STUDY

protection features to the system, so that it can tolerate server failures without loss of data. To fulfill these objectives, the first step in our approach is to characterize the system behavior, selecting the adequate KPIs. These KPIs will be used by the system manager to describe the goal policy and by the platform developers to specify the adaptations. The case study KPIs are depicted in Table II. All the KPIs were already described in Section II, with the exception of *#servers* that describes how many instances of the IMDDG/servers are active.

Next, it is necessary to translate the informal objectives into goals. The goal policy chosen for the case study is one among several, as it is possible to derive different policies that give distinct priorities to different objectives. The policy consists in three goals. The first is to maintain the redundancy, i.e., a minimum number of servers/replicas. This self-healing property is the most important goal because it will allow the system to recover from fail overs and avoid downgrading the service to a critical level. In the next goal, the system attempts to process as many requests as possible, to maximize the service ratio. Finally, the last goal minimizes the resource consumption to cut on costs, as long as other goals are not violated.

```
Goal preserve_redundancy: #servers Above 3
Goal max_service_ratio: Maximize SR MinGain 0.05 Every 300
Goal min_cost_resources: Minimize #servers Every 500
```

B. Adaptations

The platform developers have knowledge about the different settings used to configure the platform and their impacts in the behavior. In the case study, we focus on the replication degree. While Infinispan can handle changes in the number of local instances running, that cannot be adapted during runtime. The adaptation below allows to add a new server to the system and increase throughput, if the system is overloaded.

```
NodeAdaptation AddServer(n)
 Node:
  n
 Actions:
  n.addComponent(ApacheHTTP)
  n.addComponent(Infinispan)
 Requires:
  ! n.hasComponent(ApacheHTTP)
 Impacts:
  #servers += 1
  RS = (writePercentage * (1 - AR) * writeTime)^-1
 Stabilization:
  period = 120 secs
```

A new server/replica is added to an inactive node n (we have 10 nodes available in the system), which is achieved by adding the web server and Infinispan. As a result, the *#servers* KPI increases and has impact on *RS*. The impact on *RS* is specified using the average *write time*, computed by the monitor as described in [11], and the *percentage of write* requests, which is context information available in the monitor. The impact functions of RS are based on results obtained from distinct benchmarks made to the system, where different combinations of the write percentage, the key pool size, and the number of servers were explored. The impact functions used are not exact, but they provide enough accuracy for the approach.

Type	Goal	Event
Exact	*maintain_redundancy*	kpiBelow(*#servers*,3)
Approx	*maximize_SR*	kpiIncrease(*SR*,300,true)
Approx	*minimize_cost_resources*	kpiDecrease(*#servers*,500,true)

Table III
EVENTS EXTRACTED FROM THE GOALS

C. Generated Rules

With the information provided by the human operators, the proposed approach is able to generate the rules that will be used to manage the system. These rules are composed by an event (extracted from the goal) and the suitable combinations of adaptations to address the issue described by the event. The extracted events are described in Table III. One rule follows:

```
When kpiDecrease(#servers,500,true)
 Adaptations: RemoveServer(1),RemoveServer(2),...
```

V. EXPERIMENTAL EVALUATION

The prototype relies in Infinispan 5.0.0, whose local instances were deployed using the replication topology for fault-tolerance purposes. The load imposed by the web servers is emulated by the benchmark Radargun 1.1.0 [12]. The benchmark simulates the clients, the virtual server load balancer, and the web servers at each node. The benchmark detects when a new server is added to the cluster, through a monitoring agent present in each node, which notifies the virtual server's load balancer. Radargun emulates the operation of the web server, using a configurable load profile, that includes the write percentage and the object pool. The benchmark was extended to allow a finer control of the workload and the duration of each experiment. The autonomic controller was implemented in the Java™ language. The testbed consists of eleven machines. One hosts the autonomic controller and the others can run instances of Infinispan.

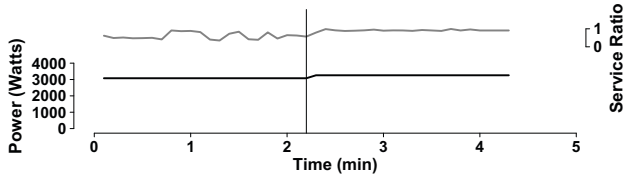
A. Workloads

We rely in several workloads to simulate variable load. The experiments have the following pattern: we first let the system stabilize in the best configuration for a given workload, then we change the workload characterization and observe how the system reacts. Changes to the workload are made such that different adaptations are more appropriate in each experiment. The workload transitions are based on 2 workloads *WL-5* and *WL-6*, which differ in terms of *load* (*RI*). The first requires 5 servers, while the second requires 6. We have experimented 2 transitions which are discussed next.

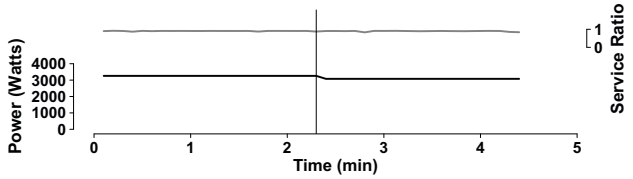
WL-5 to WL-6. The website is subject to an increase in the load. Before the increase, the system is not using total order, and requires 5 servers to process all incoming

requests. When the workload changes, the 5 servers/replicas become overloaded and the service ratio decreases. As a result, the planner selects the adaptation *AddServer* to add another web server and Infinispan instance. Figure 3(a) depicts the improvement of the service ratio after the adaptation. However, the addition of new server increases the service cost and the resource consumption, as demonstrated by the increase in the power consumption, as the machine is no longer idle. We opted to show the average power consumption because power consumption is not steady over time.

WL-6 to WL-5. This transition illustrates the inverse of the previous transition, decreasing the load, when the system is using 6 servers. At some point, the periodic event corresponding to the cost and resource consumption goal is triggered and the planner determines that it is possible to cut on costs and resources, while maintaining the service ratio. The planner selects the *RemoveServer* adaptation. Figure 3(b) shows that after the adaptation, the service ratio is maintained, power is saved and the service becomes cheaper.



(a) First scenario: adding a new instance



(b) Second scenario: removing one instance

Figure 3. Experimental results some minutes before and after the adaptation

In an extended report [13], we include results for additional adaptations, such as, a new replica update protocol to decrease the abort rate during high-contention workloads.

VI. Related Work

IMDDGs employ different technologies that several works have tried to dynamically reconfigure and tune. Many works focus on the adaptation of caching middleware. They provide adaptive solutions for the expiration time of objects in the cache [14], cache update algorithms [15], [16], or mapping of requests to groups of cache servers in web caching [17]. None of these approaches relies on high-level policies to autonomously manage the aspect they target. The planning is hard-coded in the algorithm and only takes in consideration a limited number of metrics. To extend these algorithms to consider other metrics, adaptations, and a different desired behavior, the planning would have to change significantly. In contrast, our approach provides enough flexibility to change the system's adaptive behavior, by changing the goal policy. Therefore, the self-management support does not require re-development. Nevertheless, these approaches could be integrated in our approach as part of the pool of cache policies.

VII. Final Remarks

In this paper, we present an approach for the self-management of systems employing IMDDGs, where the system manager expresses the target behavior of the middleware in terms of a high-level policy that establishes goals for a set of KPIs. The platform developers describe the possible adaptations, that are selected and deployed automatically, in response to runtime changes in the workload. The results show that the approach is able to automatically manage the system during runtime, taking advantage of the available adaptations to improve the system performance. Our previous work in centralized environments [9] shows that the planning can scale for larger sets of adaptations, which dramatically increases the difficulty of manually balancing the trade-offs necessary to specify a low-level policy, not to mention more error-prone.

Acknowledgements

This work was partially supported by the FCT (INESC-ID+LASIGE multi annual funding through the PIDDAC Program, and CMU-PT/ELE/0030/2009).

References

[1] Memcached, "See memcached.org/."
[2] Infinispan, "See www.jboss.org/infinispan."
[3] Coherence, "See www.oracle.com/."
[4] ScaleOut, "See www.scaleoutsoftware.com."
[5] Velocity, "See www.microsoft.com."
[6] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, pp. 41–50, 2003.
[7] B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, and J. Magee, Eds., *Software Engineering for Self-Adaptive Systems*. Berlin, Heidelberg: Springer-Verlag, 2009.
[8] D. Sykes, W. Heaven, J. Magee, and J. Kramer, "From goals to components: a combined approach to self-management," in *Proceedings of the International workshop on Software engineering for adaptive and self-managing systems*. New York, NY, USA: ACM, 2008, pp. 1–8.
[9] L. Rosa, L. Rodrigues, A. Lopes, M. Hiltunen, and R. Schlichting, "From local impact functions to global adaptation of service compositions," in *Proceedings of the 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 593–608.
[10] A. K. Bandara, E. C. Lupu, J. Moffett, and A. Russo, "A goal-based approach to policy refinement," *Policies for Distributed Systems and Networks, IEEE International Workshop on*, p. 229, 2004.
[11] D. Didona, S. Peluso, P. Romano, and F. Quaglia, "Self-tuning replication of elastic in-memory transactional data platforms: an analytical performance modelling approach," INESC-ID, Tech. Rep. 25, 2011.
[12] Radargun, "See sourceforge.net/apps/trac/radargun/."
[13] L. Rosa, L. Rodrigues, and A. Lopes, "Goal-oriented self-management of in-memory distributed data grid platforms," INESC-ID, Tech. Rep. 43, 2011.
[14] C. Pohl, "Adaptive caching of distributed components," PhD Thesis, TU Dresden, Dresden, Germany, 2005.
[15] N. Megiddo and D. S. Modha, "Arc: A self-tuning, low overhead replacement cache," in *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*. Berkeley, CA, USA: USENIX Association, 2003, pp. 115–130.
[16] R. Subramanian, Y. Smaragdakis, and G. Loh, "Adaptive caches: Effective shaping of cache behavior to workloads," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 385–396.
[17] S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd, and V. Jacobson, "Adaptive web caching: towards a new global caching architecture," *Comput. Netw. ISDN Syst.*, vol. 30, pp. 2169–2177, 1998.