Intelligent architecture for automatic resource allocation in computer clusters.

Sophia Corsava
Vladimir Getov
Harrow School of Computer Science

Intelligent Architecture for Automatic Resource Allocation in Computer Clusters

Sophia Corsava and Vladimir Getov,
Harrow School of Computer Science, University of Westminster, London, U.K.
Email: sophiac6@yahoo.com, V.S.Getov@westminster.ac.uk

Abstract. *As the need for more reporting and assessment of information increase exponentially, computer-based applications consume resources at an alarmingly rapid rate. Therefore, traditional techniques for managing resource allocation, topology and systems need urgent revision. In this paper, we present an intelligent architecture that introduces a new strategy for managing resource discovery, allocation and dynamic reconfiguration at run-time. Our building methodology involves the employment of new types of clustered systems based on large application groupings, each having a master cluster controller. Each controlling engine consists of self-healing intelligent entities that can compensate for a variety of software or hardware problems. We also present evaluation results of extensive experiments in a production environment, which demonstrate the advantages of our approach.*

Keywords: Cluster computing, resource allocation, resource discovery, dynamic reconfiguration.

1. Introduction

In order to achieve the expected levels of high availability, speed and service in the information society era, new methods for dynamic resource provisioning, configuration and management are urgently needed. There are currently several ways to deal with computer infrastructure availability and resource related issues [2, 3]. These include: 1) Small clusters consisting of relatively small number of nodes, that run special-purpose clustering software such as Sun Cluster, Veritas Cluster Server, HP ServiceGuard, HACMP and Siemens PrimeCluster. 2) Networks that can re-route and balance traffic, and intelligent Brocade switches with Fabric OS. 3) File system clustering, disk mirroring and high-speed WAN links amongst separate geographically sites with replication of data volumes, with products such as EMC SRDF, Timefinder, Geospan, and Veritas Volume Replicator. 4) Grids – the grid technology aims to group and present hosts, resources and services to end users [15]. 5) IBM's Océano project [14] – Océano's aim is to develop middleware and infrastructure, which provide composition of hosting services, including monitoring of service level agreements, dynamic resource allocation, and high availability.

All commercial high-availability and fault-tolerance solutions, while largely effective if implemented correctly, have the following disadvantages; 1) They can only be applied on certain types of platforms such as Sun, HP-UX, AIX, EMC, CISCO and others that are homogeneous and usually proprietary, but not on heterogeneous ones. 2) They are extremely expensive to maintain and require highly specialized personnel to support them. 3) Each one individually, cannot guarantee 100% uptime as it does not cover all aspects of the system, and if combined they introduce a substantial degree of complexity. When a type of fail-over occurs this usually means that applications crash in approximately 90% of the times. 4) They are fault-tolerant but not self-healing and most of the times non-secure. Recovery procedures require heavy manual intervention, as they are not fully automated. 5) They are not transparent to the application and very often, high availability technologies fail first and then applications subsequently crash. These solutions require expensive and powerful servers and are not usually implemented on workstations or low specification systems. Very rarely we see distributed and parallel processing architectures due to the costs incurred and the inability of most of these methods to deal with the latter without additional software. 6) They do not look after application interdependencies on a large scale.

Closely related to our project is also the very important work done by John Wilkes and R. Golding on self-managing, self-configuring storage [5, 13].

This paper is organised as follows. In section 2 we discuss the intelligent infrastructure architecture. In section 3 we focus on the strategy used by the intelligent software to discover, allocate and manage resources, while section 4 presents some experimental results from our actual implementation in a production environment.

0-7695-1926-1/03/$17.00 (C) 2003 IEEE
Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)

2. Intelligent Infrastructure Overview

2.1 Intelligent Agents

Intelligent agents [12], or intelliagents are Unix programs that monitor systems and services and wherever possible automatically correct run-time operational faults with as little downtime as possible. They can be thought of as huge wrappers that can be used to administer, maintain and troubleshoot every single infrastructure aspect. They are highly modular and use constraint-based causal reasoning to decide the best course of action [7]. A causal model is a triple that encodes the truth-values of sentences that deal with causal relationships. It includes 1) action sentences such as A will be true if we do B, 2) counterfactuals such as A would have been different if it were not for B and 3) plain causal utterance such as A may cause B OR B occurred because of A.

Intelliagents are installed locally on each server they monitor, always in the same physical location *"/apps/intelliagents"* and are "awakened" by the local Unix crons every *X* minutes (every 5 minutes for example). Intelliagents do not use a relational database (to avoid corruptions and for simplicity), they use static ontologies in the form of static knowledge templates and service lists to generate dynamic ones. Ontologies are being used in logic, mathematics and artificial intelligence. An ontology is a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents [4]. The subject of *ontology* is the study of the *categories* of things that exist or may exist in a domain [10]. Service lists are used as indexes to organise the knowledge they have for all servers, resources and services in the datacentre. In our experiments, the fastest way to create these types of ontologies was to do so manually the first time a new resource, server or service are installed and introduced to the datacentre. Whenever local intelliagents run they produce a dynamic ontology in the form of dynamic local service profiles that indicate if servers, services and resources are available for use or not.

Our intelliagents are mainly developed in bourne shell and are as likely to fail as any standard system startup script (in Unix based systems most startup/shutdown scripts are written in bourne shell) [9]. They use Unix IPC and exit codes to communicate with the operating system. The reason why we used Unix shell is because it is very easy and fast to write scripts, change them when needed or troubleshoot them. Unix pipes produce data streams and the multitude of standard Unix tools provides the core set of operators. They can produce flat ASCII data streams, which means that all operators can read each other's data. This allows for intermediate storage of output results on disk and the ability to feed them back to a process at a latter stage. Application binaries can be called and used directly. Many applications have tools that can be manually used to troubleshoot them or check their status. We took advantage of these features to ensure that intelliagents were as robust and easy to handle as possible. In addition, we did not have to install additional compilers that would put more load to monitored systems, or that would compromise security in any way. Finally, from our experience, we know that when a system is failing or is overloaded, complicated measurement/troubleshooting tools tend to stop working altogether [3]. The likelihood that a bourne shell based script would stop working, in such cases, is much less [9]. For Windows and other operating systems we used Unix shell simulations that we installed, in order to maintain uniformity in our source code.

Intelliagents use 2-phase locking which is a programming discipline that shows that no lock can be released, before the last lock has been obtained. This avoids them operating inconsistently. All actions are logged and every time an intelliagent observes a problem and takes an action, a message is sent to human operators (usually by email). Local intelliagent source code is read only. On each server a full intelliagent suite is running locally. To monitor, support and maintain intelliagents, we used external administration checkpoints that were initiated by specially built dedicated administration servers. On each external administration server intelliagent originals are kept in a secure location. Human administrators are allowed very limited access to administration servers. Access control procedures are in place to ensure that no modifications take place without detection. Daily password ageing is enforced that obliges users and administrators to change them on a daily basis. The SSH [1] protocol is used, while all other connection methods (such as remsh, rlogin, rsh, telnet, ftp) are disabled by default during the server building process.

All intelliagent related communication goes through a private agent network to avoid putting any performance/load overheads to public LANs. All participating devices and resources in the datacentre are connected to the private agent network and one or more public LANs. If the private network fails, intelliagents can automatically re-route their communication traffic over the public LAN, using Unix administration commands.

Whenever local intelliagents run, they produce flags in the dedicated *"/logs/intelliagents/intelliagent_name"* directory on the local server disk to show the status of the run. A number of flags are produced with appropriate naming conventions that show what happened and exactly where agents found a fault. Absence of these flags means that we either have an internal intelliagent problem or that they did not run at all. Administration servers monitor the creation of these flags every *X+5* minutes, where *X* is the

frequency intelliagents run, i.e. every 10 minutes (adjustable parameter). If these flags are not there, they start troubleshooting intelliagent processes. Whenever an agent detects an error it tries to fix it. All intelliagents run in parallel, in a distributed manner and do not depend on each other. At start-up each intelliagent checks to see if any other of the same type is running, if so it exits, i.e. we can never have two backup intelliagents running at the same time. It also removes flags from previous runs and old status profiles. For each application type there are customised error categories. Application health is determined by attempting to connect to them every *Y* minutes and run basic commands (such as a get on a web server process for example). This is essentially the way intelliagents communicate with applications – by trying to use them and by examining exit codes in the Unix shell.

Each intelliagent has 5 major parts: a) Monitoring, b) Diagnosing, c) Self-Healing/Action/Repair, d) Communication/Logging, and e) Self-maintenance. The monitoring part is tasked to look after one particular system resource or aspect. Whenever the monitored subject does not respond as expected, the diagnosing part is invoked and goes through a series of tests to determine the root of the problem. The diagnostic procedure is done in two ways, statically and dynamically. Statically, from parsing and examining error logs and dynamically by the use of Unix administration commands to ensure the best possible diagnosis. Based on these findings the self-healing portion gets activated and starts repairing the faults. The communication part is responsible for communicating with other intelliagents and human operators. It is also responsible for logging all intelliagent activities and results.

Intelliagents are classified based on their functions and tasks as follows: 1) Hardware agents that look after hardware components (CPU, memory, boards, etc.). 2) Operating system/network agents that look after OS and network related aspects. 3) Resource intelliagents that are responsible for managing and configuring resources such as disks, network cards, and virtual memory. 4) Application/Service intelliagents that manage and troubleshoot local and global applications/services across the data centre. 5) Status intelliagents that dynamically generate profiles about the availability, load, capacity and geographical location of servers, resources, and services.

2.2 Clusters

Clusters present a single image to the computing environment [11]. One way of achieving this is by the use of a *"heartbeat"* [11], which is commonly transmitted via ethernet crossover cables or network hubs. Usually, the size of fully-fledged production clusters in the market is relatively small with less than 10 server nodes (2 to 8). Commercial cluster software is consisted of the cluster agent software and kernel modules that integrate it to the operating system.

The cluster agent has 4 main components – the monitor that looks after the cluster, the cluster startup script, the cluster stop script and the cluster clean script that stops cleanly the cluster software. When the cluster software detects a local problem, it decides based on pre-scripted conditions if the problem is critical or not. Critical problems are classified as all conditions that do not allow the clustered component to be accessible and readily available. All other conditions are classified as non-critical. If a problem is non-critical the software takes no action. If it is critical, it initiates a failover to the next healthy available cluster node that participates in the cluster configuration. A failover is an operation by which common resources (such as disk sets) and individual services are switched over automatically to a secondary system that is identical to the failed primary one.

One or more floating IP addresses or Virtual IP addresses (VIP) are used for addressing clustered resources [11]. Clusters are inherently fault-tolerant by design as their primary function is to ensure service continuity and high availability. However they cannot dynamically resolve performance related problems or allocate resources that they do not know of. In our implementations we used the Veritas Cluster Server suite [11], as it is one of most reliable and flexible ones.

2.3 Physical Infrastructure

To prepare the physical infrastructure for the intelligent software, the cluster hardware infrastructure must be in place initially. This requires two network hubs per network segment so the private cluster network is set-up for the cluster heartbeat to travel through it [11]. All devices participating in the datacentre need to have the ability to connect to the network. Each device must have ideally 4 to 5 network interface interfaces. One is used for the public network, 2 for the cluster heartbeat and one interface for the agent/administration network. If one network breaks down for any reason, traffic can automatically be rerouted to the network part that works [11].

Three networks need to be in place for the intelligent infrastructure to work, the public network whether that is a Wide Area Network (WAN) or a Local Area Network (LAN), the cluster private network and the administration/agent network. All devices are clustered per function and operating system type. Disks, tape drives, printers, scanners, CD towers etc, are clustered wherever possible, and addressed by service names and virtual IP addresses (VIPs). All devices in each cluster group have access to all application physical and logical resources. In this way, any server can take over any

application function with minimal service interruption or none at all. In cluster configurations where applications allow for parallel processing, the common pool of resources is accessed simultaneously by all cluster nodes. In such cases, the loss of a server node does not result in service loss or downtime. Performance may be affected, however, if the application is heavily used at the time.

Figure 1 gives a general overview of our intelligent architecture using an example real implementation. Each application group is clustered with many devices in each cluster. By addressing *"oracle.database.billing.com"* one is in effect addressing all devices in that group, as the VIP can be thought of as the "index key" giving access to a whole list of aliased IP addresses corresponding to physical device NICs. All data reside on commonly shared NFS disks, and are accessible by all cluster groups. A node has one or more physical IP addresses and more than one Virtual IP addresses can be aliased to its physical IP addresses. For example, a service name such as "*email.services.com"* is associated with the virtual IP address of each cluster group, which makes addressing for humans much easier [11]. In addition, if addressing is done by service name, changing IP addresses for any reason – fault or network expansion – is much easier. Note that the service name can be anything. In this example, we follow Internet naming conventions, as they tend to be easier to remember. For each resource type, there is a corresponding *"spare.device.services.com"* pool. In these types of pools there are spare resources (non-clustered) that can be allocated when needed.

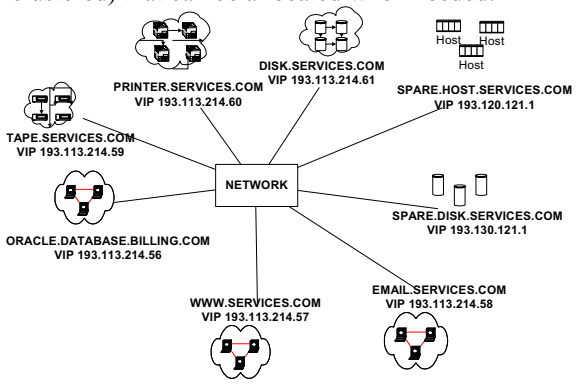

Figure 1. Example architecture for the intelligent infrastructure.

Resource discovery is made easy with this type of addressing. As all available devices are clustered, we only need to know the virtual name and not specific hardware information about each and every device. If we need for some reason to replace an entire device or parts of it, it is very easy to do so, as there are always other devices in the cluster that can be addressed by the same alias name and VIP.

Introducing new devices to the infrastructure is equally easy. Each new device is introduced as a new cluster node to the corresponding device pool it belongs. Intelliagents handle this task automatically and transparently with no service interruption as soon as the hardware connection to the private cluster network is established. To enhance redundancy, clustered devices need to have as many members as possible in the cluster. When more nodes participate in a cluster, the probability of failure decreases dramatically. Experiments [8] have shown that reliability in a multi-node cluster system increases even if the components it is consisted of are not very reliable themselves.

2.4 Interfacing clusters with heterogeneous operating systems

In our experiments we have used clusters running three different operating systems – Unix, Linux and Windows. Commercial cluster software does not work for heterogeneous operating systems. In order to overcome this limitation, we added modules to the cluster intelliagent to simulate parts of the core cluster function. As our intelliagents were developed using native operating system tools, simulating this functionality was not difficult. To this end, we have used the NFS protocol to access shared disk devices where databases and other application data were located. To create local shares if required, amongst Windows, Unix or Linux clusters, we used a utility called SAMBA [6]. SAMBA allows for an NFS share interface to be created on a server running Unix, Linux or Windows. This means that a Windows server, for example, can access a file system on a Unix server and vice versa. All kernels for all cluster groupings have the same abilities to run all clustered components and applications.

When a cluster group has a catastrophic failure and none of its members could function, cluster intelliagents would go through the following steps: 1) Shutdown all application and cluster software on the failed or failing cluster if appropriate. 2) Probe to find out which cluster has the capacity to run the failed application, based on performance thresholds and utilisation. 3) On the new cluster group, they attempted to create and bring the application group online with administration commands - dynamically without service interruption. 4) Finally they inform the human operators about what happened. (If this procedure could not be carried out by the failed hosts of the cluster, administration servers are to be used instead).

All data are located on commonly shared disk-sets over the network, so there is no problem in accessing them from various cluster groups. There are slight delays in this design, as communications and status exchange are carried out over the agent network. The reason we could

not use the cluster private network is due to the different operating systems of the cluster groups. The total time it takes to successfully create and bring online a new application group on a cluster group with a different operating system after a catastrophic failure is about 15 minutes. If intelliagents fail for any reason, human intervention is requested.

3. Resource Intelliagents

Resources in the intelligent infrastructure are automatically assimilated when connected to the network. Our design is based on the fact that for each device type there is a unique MAC address that declares the manufacturer and device type. A text file present on all servers contains a list of MAC addresses that correspond to various suppliers. The owner of this file is the administration server, which is also responsible for distributing the file every time there is a change. An IP scanner utility (initiated by administration servers) scans the network and builds a dynamic ARP table. MAC addresses and IP addresses are extracted from that table. Agents can then determine what resource type they have "discovered", by matching the discovered MAC address against those in the static list.

Cluster intelliagents probe the private cluster network every five minutes (adjustable parameter) to determine if there are any new members. The appropriate cluster group configuration is automatically downloaded to the new member from the acting cluster master. For each resource type there is a special intelliagent that holds all knowledge about it. All native operating system commands and/or specialised software that manage the resource are included in the intelliagent source control code. As all resources are clustered and can be accessed by a virtual IP address or a service name as previously discussed, resource intelliagents, do not have to waste time probing for new resources all over the place. They use "restricted" probing instead, saving time and computational resources. For example, they know that if they need resources from the disk pool, they must address the "*disk.services.com"* pool and spare space on an already used disk can be found there. If they need a completely unused disk, they can address the "*spare.disks.services.com*" pool. There are three main reasons why resource intelliagents are invoked: 1) Performance problems, 2) Capacity problems, and 3) Faults.

Performance problems can occur on a component, device or entire cluster basis. If the entire cluster needs additional processing power, a new cluster node is added to it. If a device has a performance problem, another intelliagent, the performance intelliagent, assesses the problem and decides what type of resource(s) need to be added or locally freed. Based on this assessment resource agents act. If there is a failure, they probe for spare resources in the common pool and may either come up with unused ones or allocate partially used resources. An example of a performance problem, is when database disks under-perform. If performance problems have been detected before for the same device there may be two main reasons: either there is a hardware fault that needs to be looked into, or these device(s) are not configured for the load they have been sustaining. In such a scenario, resource intelliagents reconfigure the disk layout e.g. add more disk stripes.

A joint performance/capacity problem, where these intelliagents are often called to operate, is the freeing of local "badly" used resources. In servers very often resources are consumed without being actually needed, usually because of coding errors, failed interactions with other devices or inter-process communication failures. Typical examples of these are hung network connections the server allows to occupy its network bandwidth and runaway dead or zombie processes in the kernel process table. Decisions about local resource reconfiguration, tuning and freeing, need to be very well informed. Resource intelliagents, call upon performance intelliagents to decide what to do in such cases. Performance intelliagents, contain thresholds about server, network and application performance. These thresholds are coded in the agent ontologies. Using native operating system tools, agents can decide if processes are healthy or not. Very precise mechanisms are used to determine the real cause of the performance problem. When a decision is reached, resource intelliagents remove these processes or connections forcefully and free local system resources.

As previously discussed, when the entire cluster group suffers from performance problems, an additional node is introduced to it. This node can be taken from the spare pool of host resources or from another cluster if it is not used at the time. If there is no cluster node as such, the server that is less busy is selected for this task. The threshold by which a cluster node is considered "free" is if CPU utilisation, memory, I/O and network are at 1% on average. The least busy server is determined by selecting the node that has the smallest CPU, memory, I/O and network utilisations in a cluster group, with a ceiling of 25% load. If none of these conditions are met, a request for a new node installation is communicated to the human operators.

Let us suppose that, during a backup session, there are not enough tape drives to complete it successfully. The appropriate resource intelliagent will initially contact the "*tape.services.com"* group in an attempt to find if there is a spare tape device or a device that is not used at the time. If it succeeds in finding a spare resource, it then allocates it to the requesting agent. If it finds a device that is not used at that point in time, it temporarily allocates it to the

requesting agent and a time-share is arranged for the use of this device. It invokes the backup intelligent, which holds all knowledge about the backup schedules using this device which are rearranged to avoid conflicts. If, however, it does not succeed in either case, it contacts the spare pool of tape resources. If it finds an unused one it invokes the cluster intelligent which adds the tape device to the tape cluster group. It then allocates the tape device as requested. If there are no spare backup devices to be allocated, an alert is sent to human operators asking for a new tape device to be installed to the common pool of tape devices. The cluster intelligent will then automatically incorporate this new device to the existing cluster configuration. The next time frame resource intelligents run, they use the newly added tape device.

If there are faults, the resource intelligent is called to allocate replacement resources. In such cases, the intelligent will probe for a usable resource and then call the intelligent module that will initialise the resource and introduce it to the cluster group. For the sake of this example, let us suppose that a disk has failed. If the disk is clustered, the clustered counterpart is used; if not, all accesses to that disk are disabled and cluster intelligents are called to fail-over/stop the application(s) that are using it locally (note that all applications are clustered and therefore controlled by cluster intelligents). Global interdependencies are checked and all dependant applications are stopped if necessary, while human operators are informed. Agents probe for a disk with the same size in the "*spare.disk.services.com"* pool. If this is fruitless, the "*disk.services.com"* pool is contacted to determine if there is spare space on an already used disk that can be used. If that fails too, the entire network is IP scanned and discovered IP addresses are matched against the all manufacturers' MAC address database. Intelligents connect to the discovered device (if one is found) and confirm its type using standard administration commands such as "*format*", "*ls*", etc. If an available disk is discovered, then all is well. Otherwise human operators are asked to install additional disks. If a new disk needs to be installed, the agents wait for the physical install to take place. Resource intelligents run every *X* pre-scripted time intervals and every time they check anew if the resource is present. This approach is so computationally inexpensive that has no impact whatsoever on performance or network/resource utilisation. When the new disk is detected, the appropriate intelligent disk module is called, to initialise the disk and add it to the appropriate disk group. This done successfully, any file systems are configured and mounted. The backup intelligent is called to restore data on that disk (based on the file systems the disk had) or, if the disk is clustered, it starts synchronising the replacement disk from existing data. The disk then becomes online and available. After that, the cluster intelligent restarts any stopped applications and informs in detail all human operators. At this point, interdependence and health checks resume as normal. If at any point in this procedure, intelligents fail, human operators are notified to take manual action. Figure 2 shows how the resource allocation intelligent functions. When a request is initiated by a hardware, software, performance or reconfiguration incident the resource allocation intelligent is invoked. If it is a cluster performance problem the resource allocation intelligent invokes the cluster intelligent which dynamically adds a new server node to the cluster. If it is a capacity problem, additional resources are added (for example disks) or a resource reallocation occurs by freeing used ones. The alerting and reporting intelligents ensure that human operators are kept informed about everything.

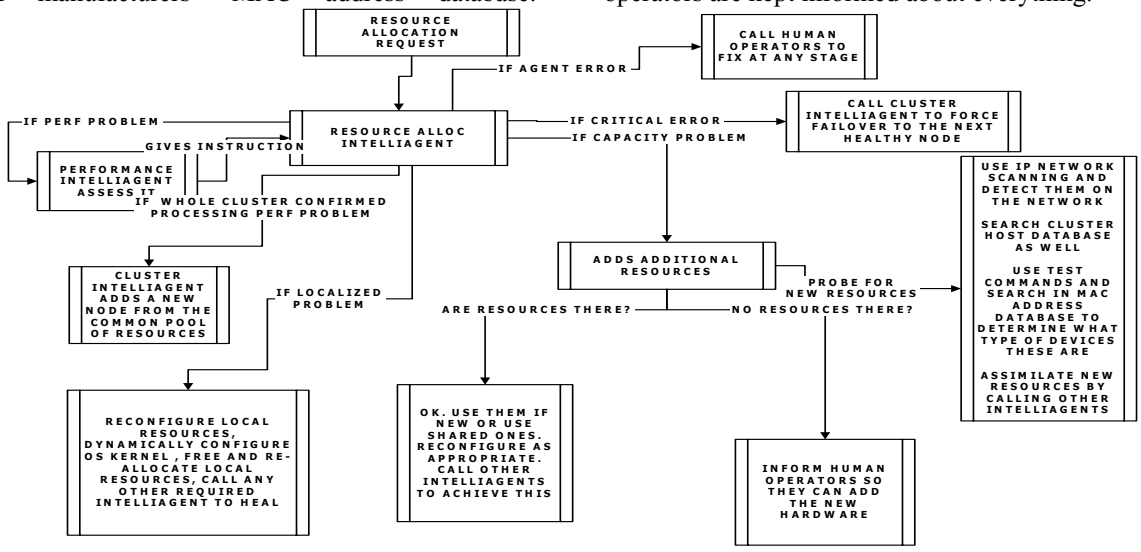

Figure 2. Resource Intelligent behaviour. Intelligents work synergistically together to allocate and manage resources

4. Experimental Results

Our production experiments were completed in a data centre consisting of about 650 servers - a mixture of Sun, HP, Linux, and Windows machines. Before the beginning of our project, all applications were running on standalone servers and data resided on local disks. Applications included web servers, databases, billing solutions, online internet shops, WAP services, chat rooms, unified messaging solutions with fax, email, and voice mail mailboxes, standard email, instant messaging and GPRS. Cluster sizes varied, ranging from 2 nodes to 150 nodes. They were both in symmetric and asymmetric configurations [11]. Most data resided on commonly shared NFS disks as per the NAS architecture [5, 13]. When we started work, we rearranged all servers in cluster groups based on operating system and functionality. We had most data moved from local disks to commonly shared NFS disks. Total testing period was 16 months with the intelligent infrastructure fully operational and 16 months without it. Faults included failing CPUs, disk corruption resulting in major database failures, database log file systems getting full and causing the database to stop transaction-logging and therefore preventing user connections, user mistakes, network failures, etc.

As discussed, prior to our work, all applications run on standalone servers so it was impossible to fail over to another healthy node, or salvage any data when the server failed completely. All efforts were concentrated on initially bringing up the server and starting the application in order to resume normal operation. In some cases a server restart would clear all problems and resulting downtime was on average 2 hours as all applications depending on the failed one had to be manually restarted. In addition to the faults, management had to be involved in order to decide on a course of action. While the actual server restart would take about 20 minutes, the whole process of locating and informing before-hand all IT and managerial resources and asking them to decide what to do would take much longer – even up to 4 hours each time. Such situations however, are totally unacceptable in production environments where time is money.

When our architecture was implemented on the same systems, clusters with more than 2 nodes exhibited 100% uptime for the entire 16 months of testing. A clustered system consisting of 2 nodes and running a heavily used production database supported 5,000 concurrent connections without any downtime or performance problems. (The same one that was unavailable for 61 hours the previous year). In the cases of failed disks hosts automatically used the clustered disk-partner without any service interruption.

On another occasion, where the disk capacity was exceeded intelliagents attempted to clear the filesystem initially by removing unnecessary files – the capacity however, was still low, so they proceeded to find available disks, brought them on line and extended automatically the file-system that was full.

Intelliagents were also used to test the application software responsiveness and health, by connecting to them and running simple test commands. If it was determined that the problem occurred because of an application malfunction they attempted to cleanly restart the entire or parts of the failed/failing application together with all dependent applications.

Figure 3 shows the significant reduction of the total downtime after the deployment of our intelligent infrastructure. From 2,172 hours of total downtime the previous 16 months, we had only 8 hours of downtime in total during the next 16 months.

As all monitoring and remedial procedures were fully automated, no time was spent trying to decide how to obtain the resource, which commands to use to initialise it, how to introduce it to the cluster configuration and how to configure it. In addition, there were no delays trying to locate experts from various fields so they could decide what exactly was the problem and which type of resource was needed (this happens mostly in performance bottlenecks and Byzantine errors). All application interdependencies were automatically re-adjusted and there were no unfortunate omissions or mistakes.

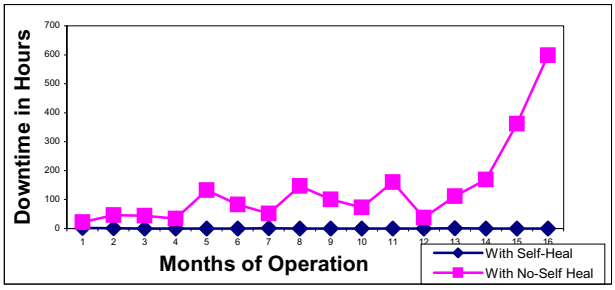

Figure 3. Intelligent Infrastructure: Total Downtime from all types of failures before and after the intelliagents.

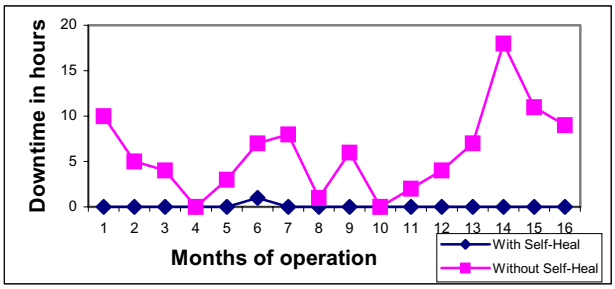

Figure 4. Intelligent Infrastructure: Downtime from resource re-allocation.

The results in Figure 4 demonstrate that only 1 hour of downtime was caused because of resource related

incidents throughout the 16 months intelliagents were operating as opposed to 273 hours during the previous 16 months without the intelliagents installed. In fact, this short downtime was caused by delays in physical resource installation, when human operators had to add a new disk for capacity reasons in month 6 of the experimental period.

5. Conclusion and Future Directions

The proposed intelligent architecture has the following new and/or specific characteristics: 1) It is not expensive to implement as it is based on existing hardware and readily available, tested technologies. In our infrastructure, we have implemented a great deal of human trouble-shooting techniques based on extensive working experience in high-end commercial environments – all actions the software takes simulate the human system administrator ones. 2) It provides very good scalability that extends the clustering paradigm to high-end data centres. Any server of any hardware configuration and specification can automatically assume any role if it is available and has access to a common pool of resources [5, 13]. 3) It manages application inter-dependencies on a large scale at the platform level. 4) It recycles and allocates various resources to running systems if needed, with minimal service interruption, or none at all [11]. 5) It is highly modular and the self-healing/recovery logic can be applied to the vast majority of operating system platforms. The software can be written in any programming language that is appropriate for the platform it runs on. New modules can be integrated very easily in the engine if and when needed. 6) It increases dramatically the availability of services to the end users and decreases the complexity of computer resource utilisation.

Much work remains to be done in the automated resource allocation and discovery areas. The design we have employed has been effective in tightly controlled architectures, but not equally effective in other non-structured environments. Additional work needs to be done so that our agents can become more adaptable in such environments. Our future work includes using hidden Markov models in the resource discovery/allocation areas and decision process. Our infrastructure design has been proven successful for the continuation of healthy, fast, and cost-effective operations and constitutes the core of a *self-healing intelligent cluster*. Any type or class of application can be supported in this model in any environment, such as databases, web servers, Internet, security, banking, telecommunications, and robotics.

References

1. Barrett, Daniel J., Silverman Richard, "SSH, The Secure Shell: The Definitive Guide", O'Reilly & Associates, February 2001.
2. Corsava Sophia, Getov Vladimir, "Self-Healing Intelligent Infrastructure for Computational Clusters", Proceedings of SHAMAN Workshop at 16th ACM ICS, New York, June 2002.
3. Corsava Sophia, Getov Vladimir, "Intelligent Fault-Tolerant architecture for cluster computing", to appear at IASTED, PDCN03, Innsbruck, Austria, Feb 2003, ACTA Press.
4. Gruber, T.A. "A Translation Approach to Portable Ontology Specifications", 1993.
5. Golding Richard, Borowsky, Elizabeth, "Fault-tolerant replication management in large-scale distributed storage systems", Proceedings 18th IEEE Symposium on Reliable Distributed Systems, 1999.
6. Hal Stern, Mike Eisler, Ricardo Labiaga, "Managing NFS and NIS", O'Reilly & Associates, August 2001.
7. Pearl Judea, "Reasoning with cause and effect", IJCAI Award Lecture, 1999.
8. Papoulis, "A. Probability, Random Variables, and Stochastic Processes", 2nd ed. New York: McGraw-Hill, 1984.
9. Quigley, Ellie, "Unix Shells by example", Prentice Hall, 1999.
10. Sowa, John F., "Knowledge Representation: Logical, Philosophical, and Computational Foundations", Brooks Cole Publishing Co, 2000.
11. Veritas Cluster Server, release 1.3.0, Veritas Software Corporation, 2000.
12. Weiss G. "Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence", MIT Press, 1999.
13. Wilkes, John and Keeton, Kimberly, "Automating data dependability", 10th ACM SIGOPS European Workshop, 2002.
14. http://www.research.ibm.com/oceanoproject
15. http://www.globus.org