Efficient Resource Management for Game Server Hosting

Dan Martin
Alpha Networks LLP
Northway, Tewkesbury, GL20 8RT, UK
dan@alpha-networks.co.uk

Graham Morgan
School of Computing Science
University of Newcastle, NE2 4NY, UK
Graham.morgan@ncl.ac.uk

Abstract

This paper describes work to ease the resource allocation problem in the domain of game server hosting. A solution was sought that required no alteration to game server code and would not inhibit a player's gaming experience. Although an academic work, the problem is tackled in a commercial setting.

1. Introduction

The standard approach to server side scalability for Internet applications is to utilise a collection of geographically co-located nodes organised into a cluster that cumulatively support online services (e.g., search engines, e-commerce, and enterprise information portals). Such nodes are standard computers in their own right, and may operate as service providers independently of each other. Such computers are general purpose and not necessarily tailored for high performance multi-processor solutions, making them a cost effective approach to server side scalability.

A common problem encountered in clustered server solutions is over provisioning. Over provisioning occurs when the amount of unused (or idle) processing resources resident in a server cluster reaches such a limit as to be considered wasteful. Each node in a cluster comes at a financial cost (e.g., maintenance, replacement) and to have more than is necessary is an overhead that should be minimised. Therefore, load balancing coupled with run-time modification of processing resources is common practice when attempting to ensure client requests are satisfied while minimising over provisioning in a cluster.

In this paper we consider a real world commercial scenario where the problem of over provisioning of servers is acute. The real world scenario is that of Alpha-Networks LLP, which maintains a cluster of over 100 nodes for the purposes of game server hosting. We describe this scenario and identify a solution that eases the problem of over provisioning without inhibiting the performance as experienced by game players. Our solution requires no modification to the existing server implementation or the application logic, and can be applied transparently. We demonstrate our approach with performance figures.

2. Background

2.1. Game Servers

In the popular First Person Shooter (FPS) genre of gaming there are companies that specialise in the provision of shared gaming scenarios for players (hosting companies). A game server is hosted within a hosting company's premises, with players accessing a game server via the Internet. Such companies are not the creators of such games or the associated servers, but are simply application hosting companies specialising in game server provision. Customers who regularly play together, and may even form a team to play others, are commonly termed clans.

Clans pay subscription rates based on the Quality of Service (QoS) delivered. Such QoS is defined in a Service Level Agreement (SLA) and may stipulate the available bandwidth, performance expectations, and availability of the game itself. Game servers are built for single node deployment only, giving the hosting company little choice but to dedicate a single node to each clan's game or installing multiple game servers on a single node (SLAs permitting). To ensure a reasonable revenue stream, in excess of one hundred nodes may be present, each running one or more game servers.

2.2. Over Provisioning

A hosting company must ensure resources are available as and when required by players. Based on a SLA, the upper limit of resource requirement is known.

However, during periods when a game server is underused or not used at all, allocation of the upper limit of resources is still required to satisfy a SLA in anticipation of game usage. This is because the game server itself is very much a ‘black box’ to the hosting company. As games of this genre are computationally expensive in terms of CPU cycles, over provisioning of the CPU is the main problem (an integral part of any game hosting SLA is CPU specification). The following example identifies over provisioning.

We assume a company has 4 nodes and SLAs with 5 clans (A, B, C, D, and E respectively). Figure 1 shows a snapshot of the current resource usage on all 4 nodes. To reduce over provisioning a simple calculation is used: if the cumulative maximum resource requirements of n clans are less than the resource availability on node x then place all these clans’ game instances on node x. To ensure the SLAs are satisfied we can see that on all nodes there are resources that will never be used (coloured black). In addition, in this current snapshot there are resources that are reserved but not used (coloured white). This occurs when less than the maximum numbers of players allowed by an SLA are participating. Only Game D appears to be utilising its full resource allocation. In an ideal scenario we could host all the existing system usage on just three nodes and save the expense of node 4.

Although the wasted amount of resources may not appear too great on just 4 nodes, we must consider that there may be in excess of over a hundred such nodes. Cumulatively, the ratio of wasted resources shown in figure 1 scaled up to a cluster of 100 nodes may be the equivalent of 25 nodes. During a typical day, only 25%-35% of resources are actually used at any one time (an even worse over provisioning problem than our example shows).

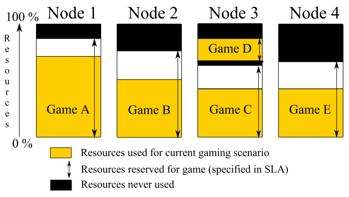

Figure 1. Typical Over Provisioning of Game Servers.

2.3. Possible Solutions

A common approach to alleviating over provisioning is via virtualisation. For the purposes of this paper we assume virtualisation describes the process of easing an application’s implementation by trivialising the use of supporting services. In particular, allowing game servers to make efficient use of a cluster of nodes in an effort to minimise over provisioning in a transparent manner. Considering different deployment scenarios, two types of virtualisation are common: (1) hiding a single node instance to provide the illusion of multiple node instances; (2) hiding multiple node instances to create the illusion of a single node.

Approach (1) allows multiple operating systems to co-exist independently of each other on different virtual nodes that share the same physical node. This has the advantage of allowing game servers that have different operating system dependencies to be co-located on the same physical node. Although this may provide more combinations to the hosting vendor for game server deployment, this approach does little to solve over provisioning: only allowing the same configuration as seen in figure 1 in terms of wasted resources. In addition, the overhead of node level virtualisation comes with additional resource overheads.

Approach (2) is commonly termed machine aggregation. A number of solutions exist, the most well known being the Parallel Virtual Machine (PVM) [3] and Message Passing Interface (MPI) [2] (in fact any standard distributed systems programming architecture may be used). Using these solutions a developer can create their applications with the ability to exploit parallel execution via associated libraries (e.g., PVM++ for C++). However, such systems are inappropriate for the hosting of game servers as the game server code is unavailable to a hosting vendor.

In principle approach (2) would solve our over provisioning problem as game servers could share resources over a cluster. However, this has to be achieved without altering the game server implementation.

3. Implementation

3.1 MOSIX

An approach that provides the benefit of approach (2), described in previous section, without the need to alter game server code may be possible via MOSIX [1]. In simple terms, MOSIX allows applications hosted on one node to utilise resources on another node as and when required in a transparent manner (e.g., when CPU usage nears 100% on host node). The unit of migration is a process in MOSIX. Figure 2 shows a ‘best case’ scenario of employing MOSIX to alleviate over provisioning in the earlier example shown in figure 1. Game servers are only installed on nodes 1 and 2, with node 3 used whenever it is required. Node 4 is unused (and may be removed completely).

MOSIX is implemented at the kernel level of the operating system (available on a number of Linux kernel versions). The decision when to migrate a process is made by MOSIX itself (using a number of resource sharing algorithms) or at a developer's/user's discretion. Each application is run on one node of a cluster (application's home node) and appears to remain there: from the user level, this migration is transparent and even those processes that are actually executing remotely are displayed as executing on the home node.

We chose openMOSIX (open source version) as our distribution for MOSIX. For our initial investigation we wanted to determine if the no-cost solution would provide benefit. There are technical differences between openMOSIX and MOSIX, but these are irrelevant in the context of this paper.

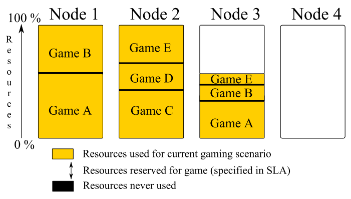

Figure 2. Desirable Provisioning.

3.2. Monitoring and Evaluation

Figure 3 provides an overview of our system. Our approach is a standard monitor, evaluate and modify approach with the ability to manually override the automated evaluation step. In addition, performance metrics in graph form may be gained via a web interface.

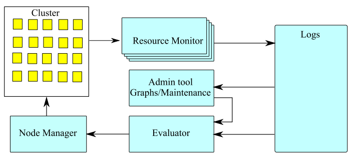

Figure 3. Overview of Resource Management Service.

Resource monitors run on each machine with the admin tool, evaluator and node manager running on a single node not included in the cluster. Programming, for the most part, was achieved using Borne Shell scripting. The only exception was the admin and graphing tool which was developed in PHP. Other technologies could have been used, but employees of Alpha-Networks LLB tend to be UNIX administrators, making our choice of technologies comfortable with staff.

The resource monitors record usage of a node (CPU in our case) in the logs. Periodically, the evaluator reads the most recent log data and determines a course of action: (1) do nothing; (2) add node to cluster; (3) remove node from cluster. The reason for adding and removing nodes is purely experimental at this stage and allows us to determine, over time, the number of nodes required to satisfy SLAs. Once a course of action has been decided, the evaluator informs the node manager. The node manager is a standard set of scripts used by administrators for manual cluster modification. However, our system now utilises such scripts without human intervention.

The evaluator can be configured a number of ways based on the number of game servers running, the different types of game servers running and any knowledge the vendor may have of pending events (e.g., online game competitions between clans). In essence, the evaluator attempts to ensure only the required numbers of nodes are available.

4. Performance

We set up an experiment consisting of 4 nodes, with 1 node having two game servers installed (similar to figure 2 without games C, D, and E). We chose nodes with a lower specification as one would usually require for supporting a game server. This was done for two reasons: (1) ensure we do exhaust CPU resources; (2) see if it is possible to provide players with the illusion of a higher specification node by combining a number of low specification nodes (financial incentive). The machines used in the cluster were PII 400 MHz-128MB RAM with Linux Redhat 9 as the operating system.

There are a number of popular game servers we tried: Counter Strike, Half Life 2, Unreal Tournament, Call of Duty, and Quake 3. Unfortunately, only one game server (Quake 3) could benefit from openMOSIX. The implementation style of an application plays an important role in determining if openMOSIX can be of benefit. Our assumption is that the threading model used in the majority of game servers is not suited to openMOSIX process migration. Excessive process handling is viewed as a performance inhibitor for real-time applications (like game servers), so avoiding their use is desirable (but not in our case). For the purposes of the initial experiment we simply want to determine the performance gain from adding and removing nodes from the cluster and if there are any nodes that are never used.

In each experiment we started the game servers (A and B) with 20 players on each (considered a high load for a game server). 19 of the players were bots (server controlled entities that act as players).

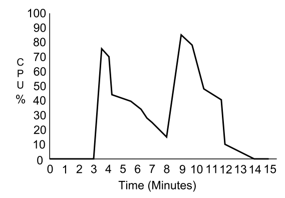

Figure 4. CPU Usage on Hosting Node.

The use of bots is common when clans wish to fill up their spare capacity, but do put additional strain on servers. We start with 1 node in the cluster (home node). We had a simple calculation to determine when to add and when to remove a node from a cluster: add node if home node CPU usage exceeds 80% remove node if home node CPU usage drops below 30%. As a safeguard to ensure resources are available, if more than one game starts at the same time on a single node 1 node is added to the cluster. In addition, if no games are running then only a single node is allowed in the cluster (home node). To ensure a change in player activity (CPU requirements) a map change event occurs on one of the game servers at 7 minutes (a new map is loaded and play is suspended for approximately 2 minutes).

All our experiments yielded similar results and figure 4 shows a graph outlining the progress of the CPU usage on the home node (taking a snapshot of a typical experimental run). At around 3 minutes the game servers start and play commences (A and B) and 1 node is added immediately (2 nodes in cluster). After 4 minutes another node is added to cope with excessive demand (3 nodes in cluster). At 7 minutes a map change event occurs in B, resulting in a node been removed from the cluster (but play continues on A, 2 nodes in cluster). At 9 minutes the new map finished loading and play continues on both game servers (A and B), requiring a node to be added (3 nodes). At 12 minutes both games finish and nodes are progressively removed from the cluster (2 nodes then 1 node).

From the graph we can see that CPU usage is never exhausted. However, from a clan's perspective the question is 'does the player witness unacceptable performance?' We judge this on the round trip time (ping) from player consoles to game server.

In the experiments all players were on the same LAN as the game servers, so ping times were not an issue due to high latency of the network (as on the Internet). Therefore, increases in ping times are due to server overloading. A simple experiment using a lightly loaded game server was used to gain the average ping time for our environment (20 - 30 ms). Repeating our initial experiment without openMOSIX players witnessed ping times rising to over 200ms (which renders a game unplayable). In fact, one of the servers became unavailable for use (players kicked off) due to excessive loads: the server is timed out by player nodes and so is not seen as an option on their consoles. With our system in place the ping times stayed within the 20 - 30 ms barrier as expected. So, not only have we utilised resources more efficiently, we have made low specification nodes provide a level of service not possible without our approach.

Node 4 was never used and may be removed permanently from the configuration (saving 1 node). For the price of three low specification nodes we are gaining the performance of a high specification node. The game servers are sharing resources more efficiently compared to single node installs.

5. Conclusion

We have demonstrated that off-the-shelf solutions coupled with simple monitoring and evaluation techniques can provide efficient automated resource management for game servers. Although only one type of game server benefited in our experiments (Quake3), it is hoped that the continual improvements to MOSIX will provide similar results for modern game servers. Therefore, it is quite conceivable that a game server hosting company may: (1) provide more game servers using the same resources; (2) provide the desired QoS with low specification machines.

Just in this limited approach, Alpha-networks LLP may limit their overheads: more clans can be supported on the existing cluster and nodes have a greater lifespan as they still provide useful processing resources after they would otherwise be viewed as outdated. In addition, SLAs may now be tailored towards gaming experience instead of node specification.

References

[1] Barak, A., and La'adan, O. "The mosix multicomputer operating system for high performance cluster computing", Future Generation Computer Systems 13, 4–5, 361–372, 1998

[2] M. P. I. Forum, "MPI: A message-passing interface standard", University of Tennessee, Tech. Rep. UT-CS-94-230, 1994

[3] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Mancheck, R., AND Sunderam, V., "PVM Parallel Virtual Machine, A User's Guide and Tutorial for Networked Parallel Computing". MIT Press, Cambridge, Mass, 1994