Task Based Load Balancing for Cloud Aware Massively Multiplayer Online Games

André Pessoa Negrão, Luís Veiga and Paulo Ferreira
Distributed Systems Group, INESC-ID
Instituto Superior Técnico, Universidade de Lisboa
andre.pessoa@tecnico.ulisboa.pt,{luis.veiga,paulo.ferreira}@inesc-id.pt

***Abstract*—In this paper we propose a task based load distribution framework for Massively Multiplayer Online Games running in hybrid cloud environments. Our solution breaks down high level tasks into subtasks in such a way that i) *core subtasks* (those with strong timing constraints) are executed at private resources owned by game operators; while ii) *background subtasks* (those with looser timing/reliability constraints) can be offloaded to temporary resources acquired from a public cloud. Our approach is lightweight and allows for faster deployment of newly acquired servers, making it more suitable for temporary overload situations. We present evaluation results confirming our solution as a viable alternative to traditional strategies.**

I. Introduction

The popularity of a Massively Multiplayer Online Game (MMOG) is hard to predict at deploy time. In addition, MMOGs have a highly dynamic active user population, which experiences variations in number even over short periods of time. These two factors combined make it difficult for MMOG operators to anticipate the exact amount of resources necessary to efficiently provision the game. As a result, operators tend to adopt pessimistic measures by deploying static and large infrastructures in which the number of resources is based in worst case predictions of load. The result is an *over-provisioned* environment in which a number of resources are idle for most of the time [1].

In light of this, several authors [1], [2], [3], [4], [5], [6] have proposed alternatives to over-provisioning based on Cloud Computing (CC). By taking advantage of the elasticity of CC [7], MMOG operators can deploy more conservative infrastructures and resort to the cloud to acquire resources according to the actual load experienced at runtime. As a result, the available resources are used more efficiently and the costs of deploying and supporting the game are reduced, benefiting every stakeholder in the MMOG environment.

While *cloud aware* resource provisioning promises to improve the cost-effectiveness of MMOGs, existing solutions employ the traditional technique of partitioning/replicating the virtual world on a fully operational game server deployed at the newly acquired resource. This approach has two main drawbacks. First, adding a new replica/partition to the system entails reconfiguration costs (e.g., data migration) at deploy time and heavy synchronization costs during the time the acquired server is up. Second, given the unreliable nature of the cloud, deploying fully operational servers might not be an acceptable solution for some MMOG companies.

978-1-5090-3216-7/16/$31.00

In this work, we propose a workload distribution solution for cloud-aware MMOGs inspired by task based computing. In our solution, when an overload event is identified, regular high level tasks that are executed by the overloaded server are broken down into subtasks that can offloaded to different resources. Tasks are partitioned so that i) subtasks with strong timing constraints or that require reliable game data are kept at the original server or offloaded to reliable resources, while ii) subtasks with less stringent timing requirements (*background tasks*) can be offloaded to temporary resources acquired from the cloud or to lightly loaded active resources.

The main advantage of this approach is that background tasks can be designed to operate over abstract data and execute low priority operations, while critical tasks/data are concentrated on reliable, long term resources. These characteristics make task partitioning more suitable for temporary overload situations (it introduces lower overhead at deploy time) and for execution in unreliable environments (it promotes the use of unreliable resources only for less critical operation).

The paper is structured as follows. Section II describes our task based load distribution framework. Section III presents implementation information. Section IV discusses the experimental results obtained. Section V overviews the related work. Finally, Section VI concludes the paper.

II. Task Partitioning

In this section we describe our task partitioning framework in detail. Without loss of generality, we discuss the process as involving two servers, the *main* (or *master*) server and the *task* (or *worker*) server. The master is the server that requests load distribution in order to have its workload lightened. The worker is the server that absorbs the extra workload coming from the master (in the form of tasks).

A. Definition and Offloading

The task partitioning process consists in breaking down a high level task that is executed as a whole at a single server into multiple tasks: one *core* subtask and one or more *background* subtasks. By default, the core task continues its execution at the main server. Background tasks, on the other hand, are meant to be offloaded to cloud resources.

The core subtask should be designed to keep responsibility for executing the most time critical operations, as well as those that require application specific information. Background tasks, on the other hand, are designed to execute operations with weaker timing and reliability requirements. In addition, background tasks should be designed to be *game-independent* – their execution should not require the installation of a copy of the game, it should depend only on abstract data (e.g., standard geometric data, such as positions and areas).

This separation of concerns between core and background tasks allows a system facing an overload situation to avoid/minimize the need for expensive code and/or data transfer. The main server retains the main responsibilities for user and data management; background tasks execute less critical operations over abstract, non-sensitive data, making them more suitable to be offloaded to unreliable resources, such as those obtained from the cloud. They can, however, also be offloaded to private resources, both new and active: because task partitioning allows the high level task to be partitioned into subtasks with arbitrarily small sizes, applications can more easily take advantage of lightly loaded active resources.

While having independent background tasks is ideal for the scenarios we envision, forcing this requirement on their design would be too restrictive. This way, application designers can also design *dependent background tasks* – background tasks that need a full copy of the game data (or a reliable partial copy). This solution gives application programmers more freedom and still allows applications to benefit from the advantages task partitioning provides in terms of separation of concerns between reliable and unreliable resources. On the downside, it has the potential of preventing the fast server loading that independent background tasks allow. A workaround to this is to design dependent subtasks with an independent *loading phase*. This way, when they are offloaded to a new server they can be executed while the application loads onto the new machine. When loading is complete, the subtask can be upgraded to its dependent phase.

B. Task Characterization

In state partitioning and state replication, the effects of adding a partition/replica are general and mostly independent of scenario: it results in a distribution of load and server↔client bandwidth at the expense of additional server↔server communication. When it comes to task partitioning, the arbitrary and application specific nature of the process means that each task partitioning strategy may have a different impact on each different resource. For example, one task partitioning policy may only reduce computational load; a different policy may have a higher impact on bandwidth, while still influencing CPU usage.

For the load distribution process to efficiently make use of task partitioning, each task partitioning strategy must have a *task descriptor* that characterizes its effects on the relevant resources. In particular, the task descriptor should describe which resources will experience a load reduction at the master and which resources will have a load increase at the worker(s). The task descriptor can also contain a quantification of these increases/reductions, although the reliability of the quantifications is difficult to ensure due to the dynamics of the execution environment.

In addition to task characterization, servers also need to be defined as *core* or *background*, in order for the system to appropriately offload tasks. By default, private resources are considered core and take precedence for the execution of core tasks. Core resources can, however, also execute background tasks if necessary.

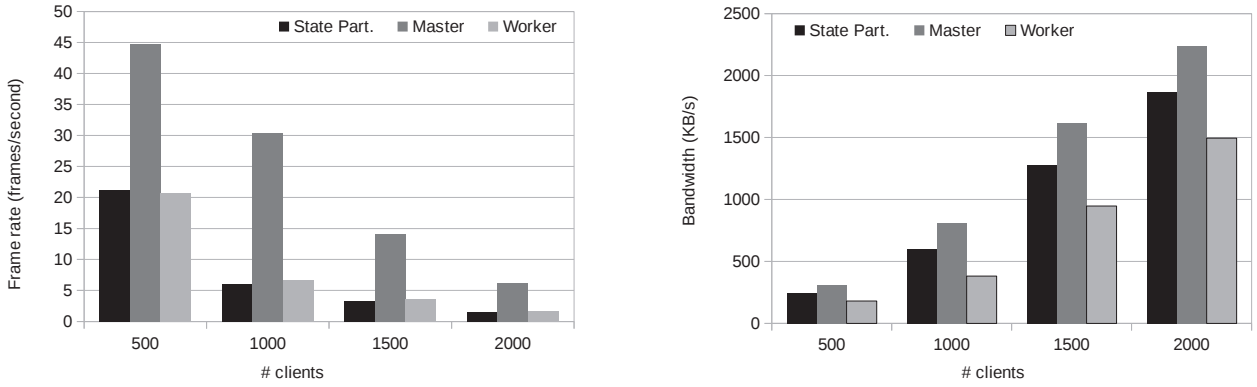
Public resources, on the other hand, are considered background by default. Individual public resources can, however, be promoted to the core group if application operators so decide. Promotion can be done temporarily *on-the-fly* (e.g., if core resources are necessary, but the system no longer has private resources available) or permanently, if operators see no reason to distinguish between public and private resources.

III. Implementation

Our task based solution is an extension to the load balancing component of CloudDReAM [8], our middleware for cost-effective execution of MMOGs in the cloud. CloudDReAM follows a hybrid cloud architecture in which virtualized servers can be acquired from public cloud providers, as well as from a private cloud run by the operators of the MMOG. To manage server→client traffic CloudDReAM uses Interest Management [9], which consists in sending to each user only updates that refer to relevant objects – typically those within an *Area of Interest* (AoI) surrounding the player's *avatar*. In particular, we use our own Vector-field Consistency (VFC) model [10], which divides the AoI into multiple zones such that: updates to objects in the inner zone are sent to the user immediately; as the distance increases, updates are propagated at increasingly lower frequencies.

Within CloudDReAM, we implemented a few default task partitioning strategies. Due to space constraints, in this paper we focus on the two main strategies: matching and update propagation partitioning. *Matching* is one of the main tasks executed by a game server. It consists in identifying, for each user, which objects are within the user's AoI and, of those, which ones have been modified (and, consequently, need to be propagated to the user). The matching process naturally lends itself to partitioning into two subtasks: the first subtask identifies which objects are within each client's AoI; the second task identifies which of those objects have been updated and need to be sent to the user.

Of these two tasks, the first one can safely be executed at a task server, which becomes responsible for identifying and informing the main server of modifications to each user's AoI. To make sure that the natural synchronization delays between the master and the worker do not result in missed interactions, the worker monitors and informs the master about each user's *extended* AoI – the actual AoI, plus an additional area surrounding it. The master can, this way, keep track of the objects in the nearby area and timely identify new additions to any AoI.



(a) Frame rate.

(b) Upload bandwidth.

Figure 1. Performance comparison between state and task partitioning.

To partition update propagation, we take advantage of the multi layered characteristics of VFC by having the task server propagating updates that refer to the less critical consistency zones. Updates regarding high priority zones continue to be managed by the master.

IV. Evaluation

In this section we analyse how our solution fares against state partitioning. To evaluate our solution, we deployed a 9 machine cloud from which new servers (VMs) are acquired as necessary. We compare the performance of the system when all servers are state partitioned against a version of the system mixing task servers and state partitioned servers. For clarity, we focus our evaluation on partitions that in one case are state partitioned and the other are task partitioned. Thus, our results show a pairwise comparison between the two strategies: a pair of partitioned servers *versus* a main and a task server.

Throughout the analysis, we show the results obtained with a varying number of clients. Clients were simulated by bots, each controlling a single avatar that explores the game map. We deployed clients evenly across the game map, resulting in approximately the same number of clients in each partition; as a result, the performance of the state partitioned server is very similar. For this reason the figures show the averages of the values obtained by each pair of state partitioned servers. By contrast, due to their inherently asymmetrical relation, we show the results of the master and worker servers separately. The number of clients presented in each figure corresponds not to the total number of clients in the system, but to the number of clients shared by each pair of servers.

Figure 1(a) shows the performance differences between state and task partitioning in terms of frame rate. *Frame rate* measures how frequently the server is able to execute the matching algorithm and send updates to clients. It is a crucial high level metric that determines if the application is able to provide its required quality of service.

As shown in the figure, the performance improvements of offloading the matching task enable the system to ease the natural degradation in frame rate that occurs as the number of clients increases. Our solution is able to achieve frame rates higher than 10 fps for up to 1500 clients, delivering optimal frame rates for slow paced games and suitable for most types of games. At 2000 clients, frame rates drop to approximately 6 fps, which, while not generally suitable, are still enough for many types of games, especially under high load. The partitioned system, on the other hand, experiences a clear sharp decrease with more than 500 clients.

The frame rates of the worker, on the other hand, are considerably lower than the master. This is due to the fact that the worker has to execute the two parts of the matching algorithm: 1) identifying if an object is within the AoI of another and, if so, 2) identifying the specific VFC consistency zone of such object. The worker, however, is responsible for propagating updates that refer to the lower priority zones, which have weaker requirements in terms of propagation frequency. Considering the values achieved, it is unlikely that the lower frame rates of the worker prevent the timely propagation of such lower priority updates.

As Figure 1(a) shows, the worker is, nevertheless, able to obtain higher frame rates than the state partitioned servers. The main reason for this is that as the number of users increases, update processing takes on a larger role at those servers, due to conflict detection and synchronization with neighbour servers, among other application specific issues. The worker, on the other hand, does not handle such issues, because it only deals with lower priority updates and does not participate in neighbour synchronization.

Figure 1(b) shows the upload bandwidth results obtained. The results show that the master is able to offload between 35%-40% of its bandwidth requirements to the worker. These values put the maximum per-server upload bandwidth requirements of the task based solution, at most, 15% above those of the state partitioned system, which has an approximately even ratio between each pair of servers.

Note that the balanced distribution between state partitioned servers only occurs because players have been evenly dis-

tributed across partitions. In practice, however, this situation is highly unlikely and differences in load between partitioned servers occur naturally. In fact, if the difference increases past a certain threshold (due to hotspots), one of the servers becomes overloaded. Overcoming this situation requires acquiring a new server or, at least, redistributing the load among active servers. In any case, this it involves expensive server reconfigurations, adding more complexity to the system.

Task partitioning, on the other hand, is less exposed to hotspots because workload partitioning between the master and the server is mainly determined by *AoI density* (i.e., the average number of objects inside a user's AoI), whereas in partitioned systems the main factor is *partition density*. Hotspots impact both of these factors, but have a stronger influence on partition density, which depends exclusively on the number of players in the partition. AoI density, on the other hand, additionally depends on user behaviour and increases at a slower rate.

V. Related Work

Nae *et al.* developed some of the most seminal work on cloud aware resource provisioning for MMOGs [11], [1]. Among their main contributions is a neural-network load prediction algorithm that identifies the resources necessary to provision a game environment over time. Marzolla *et al.* [5]propose using a Queueing Network model to balance load among servers in a multi-tier cloud infrastructure. Najaran *et al.* [2] propose renting game servers from the cloud and organize them in a P2P network. Glinka *et al.* [4] also proposed a cloud aware infrastructure, but organize servers as a replicated cluster.

The common factor between these solutions is that they are based on traditional state partitioning/replication workload distribution strategies, which have some important limitations. First, they require full game servers, which take time to become fully operational in the cloud. Load prediction may minimize this problem, but makes the system too dependent on its accuracy. Second, they involve server reconfiguration and synchronization costs that are not suitable for temporary (and, potentially, short-term) overload situations. Finally, they expose application data to the unreliable resources of the cloud, which might not be acceptable for some game companies.

Our solution tackles these issues by allowing fast deployment of servers executing background tasks, which, by design, are also more appropriate for execution in unreliable resources. In addition, the task partitioning process is more stable, as it maintains the most critical operations and data at the original server.

Lim and Lee [12] have previously proposed a task based load distribution scheme for MMOG-like environments. Their solution allocates fine-grained, message-level tasks to servers on a cluster at runtime according to CPU and network load. However, their solution assumes a static infrastructure and requires servers to be connected through high-speed networks, which contrasts with the cloud environments our work targets.

VI. Conclusions

In this paper we proposed a task-based load balancing architecture for Massively Multiplayer Online Games running in cloud environments. Our system breaks down high level server tasks into subtasks of arbitrary size and priority that can be offloaded to other servers. Critical tasks are meant to keep executing at the original server, while lower priority tasks are designed to be offloaded to public cloud resources. In comparison with the load distribution strategies typically used in MMOGs, our solution is faster to deploy (making it more suitable for temporary overload situations) and more appropriate for execution in unreliable cloud resources. In addition, it is general enough to allow applications to freely define how and which tasks should be partitioned, as well as the conditions for their offloading.

Acknowledgments: This work was partially supported by national funds through FCT – Fundação para a Ciência e Tecnologia with reference UID/CEC/50021/2013.

References

[1] V. Nae, A. Iosup, and R. Prodan, "Dynamic resource provisioning in massively multiplayer online games," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 3, pp. 380 –395, march 2011.

[2] M. T. Najaran and C. Krasic, "Scaling online games with adaptive interest management in the cloud," in *Proceedings of the 9th Annual Workshop on Network and Systems Support for Games*, ser. NetGames '10. Piscataway, NJ, USA: IEEE Press, 2010, pp. 9:1–9:6.

[3] E. Carlini, M. Coppola, and L. Ricci, "Integration of p2p and clouds to support massively multiuser virtual environments," in *Proceedings of the 9th Annual Workshop on Network and Systems Support for Games*, ser. NetGames '10. Piscataway, NJ, USA: IEEE Press, 2010, pp. 17:1–17:6.

[4] S. Gorlatch, D. Meilaender, A. Ploss, and F. Glinka, "Towards bringing real-time online applications on clouds," in *Computing, Networking and Communications (ICNC), 2012 International Conference on*, 2012, pp. 57–61.

[5] M. Marzolla, S. Ferretti, and G. D'Angelo, "Dynamic resource provisioning for cloud-based gaming infrastructures," *Comput. Entertain.*, vol. 10, no. 3, pp. 4:1–4:20, Dec. 2012.

[6] C.-F. Weng and K. Wang, "Dynamic resource allocation for mmogs in cloud computing environments," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2012 8th International*, 2012, pp. 142–146.

[7] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[8] A. P. Negrão, M. Adaixo, L. Veiga, and P. Ferreira, "On-demand resource allocation middleware for massively multiplayer online games," in *Proceedings of the 2014 IEEE 13th International Symposium on Network Computing and Applications*, ser. NCA '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 71–74.

[9] K. L. Morse, "Interest management in large-scale distributed simulations," University of California, Irvine, Department of Information and Computer Science, Technical Report ICS-TR-96-27, Jul. 1996.

[10] L. Veiga, A. Negrão, N. Santos, and P. Ferreira, "Unifying divergence bounding and locality awareness in replicated systems with vector-field consistency," *Journal of Internet Services and Applications*, vol. 1, pp. 95–115, 2010.

[11] V. Nae, R. Prodan, and T. Fahringer, "Cost-efficient hosting and load balancing of massively multiplayer online games," in *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, 2010, pp. 9–16.

[12] M. Lim and D. Lee, "A task-based load distribution scheme for multi-server-based distributed virtual environment systems," *Presence*, vol. 18, no. 1, pp. 16–38, 2009. [Online]. Available: http://dx.doi.org/10.1162/pres.18.1.16