POGGI: Puzzle-Based Online Games on Grid Infrastructures*

Alexandru Iosup

Electrical Eng., Mathematics and Computer Science Department
Delft University of Technology, Delft, The Netherlands
A.Iosup@tudelft.nl

Abstract. Massively Multiplayer Online Games (MMOGs) currently entertain millions of players daily. To keep these players online and generate revenue, MMOGs are currently relying on manually generated content such as logical challenges (puzzles). Under increased demands for personalized content from a growing community, it has become attractive to generate personalized puzzle game content automatically. In this work we investigate the automated puzzle game content generation for MMOGs on grid infrastructures. First, we characterize the requirements of this novel grid application. With long-term real traces taken from a popular MMOG we show that hundreds of thousands of players are simultaneously online during peak periods, which makes content generation a large-scale compute-intensive problem. Second, we design the POGGI architecture to support this type of application. We assess the performance of our reference implementation in a real environment by running over 200,000 tasks in a pool of over 1,600 nodes, and demonstrate that POGGI can generate commercial-quality content efficiently and robustly.

1 Introduction

Massively Multiplayer Online Games (MMOGs) have emerged in the past decade as a new type of large-scale distributed application: real-time virtual world simulations entertaining at the same time millions of players located around the world. In real deployments, the operation and maintenance of these applications includes two main components, one dealing with running the large-scale simulation, the other with populating it with content that would keep the players engaged (and paying). So far, content generation has been provided exclusively by human content designers, but the growth of the player population, the lack of scalability of the production pipeline, and the increase in the price ratio between human work and computation make this situation undesirable for the future. In this work we formulate and investigate the problem of automated content generation for MMOGs using grids.

* We gratefully thank Dr. Dick Epema and the Delft ICT Talent Grant for support, and Dr. Miron Livny for access to the experimental environment.

H. Sips, D. Epema, and H.-X. Lin (Eds.): Euro-Par 2009, LNCS 5704, pp. 390–403, 2009.

MMOGs are increasingly present in people's lives and are becoming an important economic factor. The number of players has increased exponentially over the past ten years [1], to about 25 million players at the end of 2008 [1,2]. Successful MMOGs such as World of Warcraft and Runescape number each over 3,000,000 active players.

There are many types of content present in MMOGs, from 3D objects populating the virtual worlds, to abstract challenges facing the players. The MMOG operators seek from these content types the ones that entertain and challenge players for the longest time, thus leading to revenue [3]. *Puzzle games*, that is, games in which the player is entertained by solving a logical challenge, are an important MMOG content type; for instance, players may spend hours on a chess puzzle [4,5] that enables exiting a labyrinth. Today, puzzle game content is generated by teams of human designers, which poses scalability problems to the production pipeline. While several commercial games have made use of content generation [6,7,8,9], they have all been small-scale games in terms of number of players, and did not consider the generation of puzzle game content. In contrast, in this work we address a new research topic, the large-scale generation of puzzle game instances for MMOGs on grid infrastructures.

Invented and promoted by industry, MMOGs have recently started to attract the interest of the distributed systems [10] and database [11] communities. However, these communities have focused so far on resource management for large-scale MMOGs, spurring a host of results in resource provisioning [2], scalability [12,13], etc. for the world simulation component. In this work we propose the new research direction of generating (puzzle game) content for MMOGs. Towards this end, our contribution is threefold:

1. We formulate the problem of puzzle game generation as a novel large-scale, compute-intensive application different from typical applications from grids and parallel production environments (Section 2);
2. We propose an application model (Section 3) and an architecture (Section 4) for generating puzzle games at large scale;
3. We show through experiments in a real environment that our proposed architecture can generate content of commercial quality (Section 5).

2 Problem Formulation

In this section we formulate the problem of generating puzzle content for MMOGs.

2.1 The Need for Automatic Puzzle Content Generation

Puzzle games have two attractive characteristics: they can be embedded in MMOGs such as World of Warcraft to keep online the players who prefer thinking over repetitive activities, and they are preferred over other genres by the majority of the online game players [14, p.24]. However, the current industry approach for puzzle content generation does not scale. Due to the business model associated with MMOGs, in which the content is the main factor in attracting

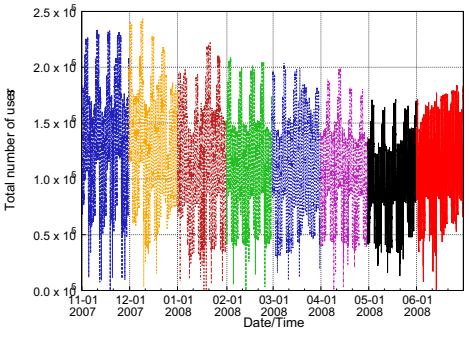

Fig. 1. The number of Active Concurrent Players in the RuneScape MMOG between Nov 1, 2007 and Jul 1, 2008. Each month is depicted with a different type of line.

and retaining paying players [3], game operators prefer with few exceptions to generate the content in-house. Thus, the current market practice is to employ large teams of (human) designers for game content generation. On major titles, teams of 50 people or more can be involved with the content creation pipeline at the same time; for increased scalability such teams are broken down into production units of several persons each under a production hierarchy, which raises the production cost. The manual content generation approach does not scale to larger teams, due to cost and time overheads. This problem is made more difficult by the impossibility to know in advance how the player population will evolve over time: for some MMOGs the population will ramp up or scale down in a matter of weeks or even days [2].

2.2 Challenges in Puzzle Game Content Generation for MMOGs

We identify two main challenges in solving the problem of puzzle content generation for MMOGs: puzzle difficulty balance and scalability to large numbers of players. Failing to meet any of these challenges has a direct impact on revenues: it leads to a degraded game experience and causes the players to leave the game [3, 15].

The MMOG players gain virtual reputation with their ability to excel in game activities such as puzzle solving. Thus, the puzzle game instances that various players solve must be matched with the player ability, and be comparable in difficulty for players of the same ability; we call this combined challenge the *puzzle difficulty balance*.

The ability to scale to the large numbers of players specific to MMOGs means that enough puzzle game instances are available for each player requiring them, so that the response time after requesting an instance is very low; we call this the *scalability* challenge. While popular MMOGs may have millions of users, only a fraction are active at the same time, the Active Concurrent Players (ACP) [3]. Thus, the number of puzzle game instances needed at any time is limited by the peak ACP volume. To understand the ACP variation over time we have collected

Table 1. Comparison between the Puzzle Generation workloads and the workloads of grid and parallel production environments (PPEs)

	Workload Type		
	Puzzle Gen.	Grids [16]	PPEs [17]
Users	100,000s	10s-100s	10s-100s
Performance metrics	HTC and HPC	HPC [18]	HPC [19]
Workload	Very Dynamic	Dynamic	Static
Response Time	Very Low	High	Very High
# of Jobs/Result	10s-1,000s	10s-100,000s	1-100s
Task coupling	Workflows	Workflows, Bags-of-Tasks	Parallel, Single

traces from RuneScape, the second-most popular MMOG by number of active players. The operators of RuneScape publish official ACP counts through a web service interface; only the ACP count for the last few seconds is available. We have collected over 8 months of ACP count samples with a 2-minute sampling interval. Figure 1 shows that for RuneScape the daily ACP peaks are between 100,000 and 250,000 players world-wide. (Peaks are several hours long.)

2.3 Puzzle Content Generation Using Grid Infrastructures

Our vision is that the puzzle game content will be automatically generated by a computing environment that can dynamically adjust to a large-scale, compute-intensive, and highly variable workload. Grid environments and to some extent even today's parallel production environments (PPEs) match well the description of this environment. However, whether these environments can support the MMOG puzzle generation workloads and their associated challenges is an open research question. Table 1 compares the characteristics of the MMOG workloads with those of the typical workloads present in these environments. Both the current grids and PPEs serve two-three orders of magnitude fewer users than needed for MMOGs. Grids and PPEs handle well dynamic and static workloads, but have problems dealing with the very dynamic workload characteristics of MMOGs (such as bursts of jobs). The response time model for grid and PPE applications permits middleware overheads higher than what is acceptable for MMOG workloads. Efficiently running in grids large workflows or large numbers of related jobs to produce a unique final result are active research topics [20, 21].

For the moment, we do not see as viable an approach in which the machines of the players are used to generate content without some form of supervision or verification. Though the machines of the active players may have enough spare cycles, as in MMOGs players compete with each other for virtual or even real-world gains, cheating can be financially profitable. Moreover, the game developers do not want to give away their content generation algorithms, which are an important competitive advantage, even in binary form. More work on distributed trust is digital rights management is needed before such an approach can be viable.

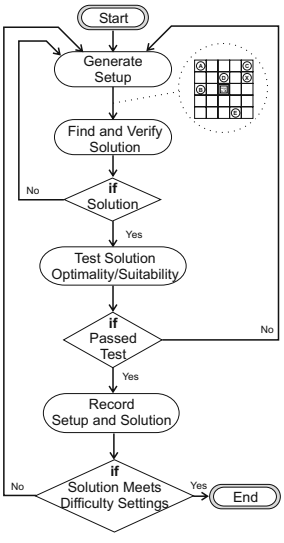

Fig. 2. The workflow structure of a generic Puzzle game instance generator

3 Application Model

In this section we model puzzle content generation as a workflow; ours is the first workflow formulation for this novel application domain.

The puzzle game generation application consists of two main functional phases: generating a solvable puzzle game instance and finding a solution for it, and testing that the proposed solution is minimal in solving effort. The second phase addresses the puzzle difficulty balance challenge (see Section 2.2), so that players cannot solve a generated puzzle in a simpler way. The two functional phases are executed until both complete successfully, or until the time allocated for their execution is exceeded. Thus, our puzzle generation application is one of the first iterative grid workflows; given the number of iterations observed in practice (see Section 5.2), it is also one of the largest.

3.1 Workflow Structure

We model the generic puzzle game generation application as the workflow with seven levels depicted in Figure 2: *Generate Setup*, *Find and Verify Solution*, *if Solution*, *Test Solution Optimality/Suitability*, *if Passed Test*, *Record Setup and Solution*, and *if Solution Meets Difficulty Settings*. The first three (the following two) levels correspond to the first (second) functional phase of the workflow; the other levels are added for output and ending purposes.

The *Generate Setup* level generates a puzzle game setup, such as a board and the pieces on it, that needs to be solved. The execution time of this level depends on what functionality it provides: this level may be designed to generate random setups including ones that break the rules of the puzzles, or include the verification of the setup validity against the rules.

The *Find and Verify Solution* level solves the puzzle starting from the setup generated in the previous level and using one of the many game solving techniques to evolve from setup to the solution state; for reviews of these techniques we refer to [22, 23, 24]. Depending on the amenability of the puzzle to different solving strategies, this level may employ algorithms from brute force search to genetic algorithms, game-tree search, and SAT-solving. Regardless of the algorithm employed, the execution time for this level is bounded by the maximum execution time set by the application designer.

The *Test Solution Optimality / Suitability* level attempts to determine if the found solution is optimal, and if it is suitable for being recorded (i.e., is the solution of the right complexity, but also of the type that will entertain the players?). Depending on the difficulty of testing these two aspects of the solution, the execution time of this level may be similar to that of the *Find and Verify Solution* level.

The *if Solution Meets Difficulty Settings* level was introduced at the end of the workflow to ensure the recording of puzzle setups whose best solutions do not meet the current difficulty settings, but have passed all the other tests and may therefore be used for lower difficulty settings.

3.2 Example: The Lunar Lockout Puzzle

We demonstrate the use of the application model introduced in the previous section by applying it to a sample puzzle game generation application: the generation of instances for the Lunar Lockout puzzle [25]. This commercial puzzle is played on a board on which pins are placed from the start and may be moved according to the following game rules to a goal position. A move consists of pushing a pin either horizontally or vertically, until it is blocked by another pin. The moved pin will be placed in the board cell just before the cell of the blocking pin, considering the direction of the push. The moved pin cannot hop over another pin or be moved outside the board. The goal is to place the X pin on a target position. A solution consists of a time-ordered set of pin movements leaving the 'X' pin on the target position. The puzzle is characterized by the size of the board N, the number of pins P, and the difficulty settings D (i.e., number of moves for an entertaining solution depending on the player's level). A *puzzle setup* consists of N, P, and the initial position of the pins; see Section 5.1 for a generated puzzle setup. The mapping of this application to a puzzle-generating workflow is described below.

The *Generate Setup* level generates a random positioning of the pins on the board, ensuring that the basic rules of the game are enforced, e.g., the pins do not overlap.

The *Find and Verify Solution* level uses backtracking to solve the puzzle. This corresponds to the situation common in practice where no faster solving algorithms are known. The choice of backtracking also ensures low memory consumption and efficient cache usage for today's processors. This workflow level stops when a solution was found.

The *Test Solution Optimality / Suitability* level also applies backtracking to find all the other solutions with a lower or equal number of movements (size). This level needs to investigate moves that were not checked by the *Find and Verify Solution* level; for backtracking this can be easily achieved with minimal memory consumption and without redundant computation. If a solution with a lower number of movements is found, it becomes the new solution and the process continues. The solution suitability test verifies that the best found solution has a minimum size that makes the puzzle interesting for its players, e.g., the size of the solution is at least 4 for beginner players.

4 The POGGI Architecture

In this section we present the Puzzle-Based Online Games on Grid Infrastructures (POGGI) architecture for puzzle game generation on grid infrastructures.

4.1 Overview

The main goal of POGGI is to meet the challenges introduced in Section 2.2. In particular, POGGI is designed to generate puzzle game content efficiently and at the scale required by MMOGs by executing large numbers of puzzle generation workflows on remote resources. We focus on three main issues:

1. Reduce execution overheads. First, by not throwing away generated content that does not meet the current but may meet future difficulty settings (see the last workflow level in Section 3.1) our architecture efficiently produces in advance content for future needs. Second, in previous work [26] we have found the job execution overhead to be an important performance bottleneck of current grid workflow engines; for this reason, no current (grid) workflow engine can be used to handle the large number of tasks required by our application (see Section 5.2). To address this problem, we have developed a workflow engine dedicated to puzzle content generation.

2. Adapt to workload variability. Using the long-term traces we have acquired from one of the Top-5 MMOGs, we have recently shown [2] that MMOGs exhibit high resource demand variability driven by a user presence pattern that changes with the season and is also fashion-driven, and by a user interaction pattern that changes with the gameplay style. We use detailed statistics of the puzzle use and of the content generation performance to adapt to workload variability.

3. Use existing middleware. The architecture uses an external Resource Management Service (RMS), e.g., Condor [27], Globus and Falkon [28], to execute reliably bags-of-tasks in which each task is a puzzle generation workflow, and to monitor the remote execution environment. The architecture is also interfacing with an external component, *Game Content*, which stores the generated content, and which collects statistical data about the actual usage of this content.

4.2 Main Components

The six main components of POGGI are depicted in Figure 3, with rectangles representing (nested) components, and arrows depicting control and data flows.

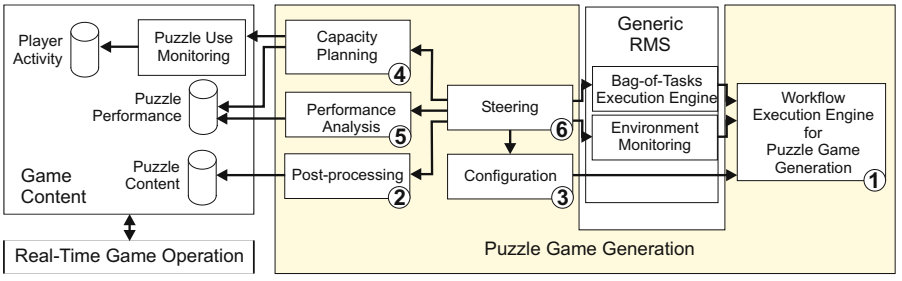

Fig. 3. Overview of the POGGI architecture for puzzle game content generation

1. Workflow Execution Engine for Puzzle Game Generation and *2. Post-Processing*: Generic workflow execution engines that can run the puzzle generation workflows (see Section 3.1) already exist [29, 30, 31, 32]. However, they are built to run the workflow tasks on remote resources through the services of the RMS, which leads to high communication and state management overheads [28, 26]. In contrast to this approach, we introduce in our architecture a specialized component for the execution of the puzzle generation workflows on a single resource; we show in Section 5 that our workflow engine can handle the large number of tasks required for puzzle content generation. While this component can be extended to execute workflow tasks on multiple resources similarly to the execution of bag-of-tasks by Falkon [28], the presence of thousands of other workflow instances already leads to high parallelism with minimal execution complexity. The *Post-Processing* component parses the workflow output and stores the generated content into a database.

3. Configuration: The generic workflow model introduced in Section 3.1 can generate content for all the difficulty levels, if the difficulty settings are set to the maximum and the suitability level is set to the minimum. Then, all the generated setups that are solvable are recorded. However, this configuration is inefficient in that high difficulty states will be explored even for beginner players. Conversely, settings that are too strict may lead to ignoring many possibly useful setups. Thus, we introduce in the architecture a component to set the appropriate configuration depending on the level of the players in need of new content.

4. Capacity Planning: As shown in Figure 1, the ACP volume reaches daily peaks of twice the long-term average. In addition, the use of a specific puzzle game may be subject to seasonal and even shorter-term changes. Thus, it would be inefficient to generate content at constant rate. Instead, the capacity planning component analyzes the use of puzzles and the performance of the content generation process and gives recommendations of the needed number of resources. Given the high number of players, enough performance analysis data is present almost from the system start-up in the historical records. We have evaluated various on-line prediction algorithms for MMOG capacity planning in our previous work [2].

5. Performance Analysis: Detailed performance analysis is required to enable the capacity planning component. The performance analysis component focuses

on two important performance analysis aspects: extracting metrics at different levels, and reporting more than just the basic statistics. We consider for performance analysis job-, operational-, application-, and service-level metrics. The first two levels comprise the traditional metrics that describe the execution of individual [19] and workflow [33] jobs. The *application-level metrics* and the *service-level* metrics are specific to the puzzle generation application. At the application-level the analysis follows the evolution of internal application counters (e.g., number of states explored) over time. At the service-level the analysis follows the generation of interesting puzzle game instances. The performance analysis component performs an in-depth statistical analysis of metrics at all levels.

6. Steering: To generate puzzle game instances, the various components of our architecture need to operate in concert. The steering component triggers the execution of each other component, and forcefully terminates the puzzle generation workflows that exceed their allotted execution time. Based on the capacity planning recommendations and using the services of the generic RMS, it executes the puzzle generation workflows.

4.3 Implementation Details

We have implemented the Steering component of our architecture on top of the GrenchMark [34, 26] grid testing tool, which can already generate and submit multi-job workloads to common grid and cluster resource management middleware such as Condor, Globus, SGE, and PBS. Thus, the POGGI architecture is not limited to a single middleware, and can already operate in many deployed environments. For this work we have extended the workload generation and management features of GrenchMark, in particular with the ability to generate the bags-of-tasks comprising puzzle generation workflows.

We have built the Performance Analysis component on top of the GrenchMark tool for analyzing workload execution, which can already extract performance metrics at job and operational levels [26]. We have added to this component the ability to extract performance metrics at the application and service levels.

5 Experimental Results

In this section we present our experimental results, which demonstrate that POGGI can be used in real conditions to generate commercial-quality content.

We have performed the experiments in the Condor pool at U.Wisconsin-Madison, which comprises over 1,600 processors. The system was shared with other users; for all experiments we have used a normal priority account.

5.1 Lunar Lockout: Solved and Extended

The commercial version of the game [25] consists of a playing set and cards describing 40 puzzle instances. These instances have been generated manually by a team of three content designers over a period of about one year. The instances can be characterized in our application model (see Section 3.2) as $N = 5$, $P =$

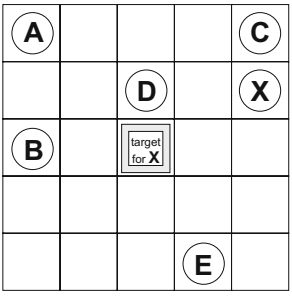

Fig. 4. Lunar Lockout instance

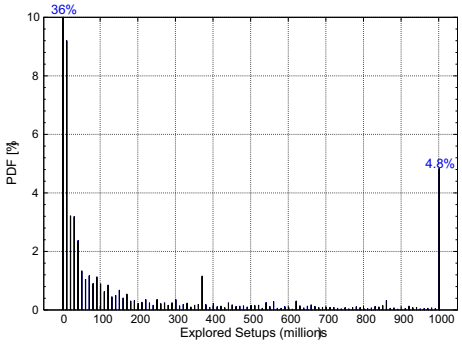

Fig. 5. The PDF of the number of explored puzzle setups per workflow. Each bar represents a range of 10 million puzzle setups.

$4-6$, and D such that the solution size ranges from 4 (beginner player) to 10 (advanced). During our experiments we have generated and solved all the boards that come with the commercial version of the game. This demonstrates that our architecture can be used to produce commercial-quality content much faster than by using the manual approach.

In addition, we have generated automatically many new puzzle instances with equal or larger boards ($N > 5$), more pins ($P > 6$), and solutions of up to 21 moves, corresponding to an expert play level that exceeds the human design capabilities. Figure 4 depicts a sample setup that has been automatically generated for $N = 5$, $P = 6$, and $D = \{$ *solution size* $\geq$ *8 for advanced players*, *solution size* $\geq$ *4 for beginner players*, $\ldots\}$. It turns out that the best solution for moving the X pin to the target has 8 moves: A→Right, A→Down, X→Left, A→Up, E→Up, A→Left, X→Down, and X→Left.

5.2 Application Characterization

To characterize the puzzle generation application we use a test workload comprising 10,000 workflows. Overall, the execution of this workload led to the

Table 2. The number of interesting states found for each 1,000 jobs, per solution size.

Configuration	Solution Size																
	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Normal	1,281	1,286	645	375	230	134	75	47	27	11	6	1	2	-	-	-	-
Large	-	-	409	334	257	147	171	57	79	83	39	41	24	22	2	3	7

evaluation of 1,364 billion of puzzle setups (tasks). The number of tasks is much larger for puzzle content generation than for the largest scientific workflows; the latter comprise rarely over a million of tasks [20].

The probability distribution function (PDF) of the number of explored puzzle setups for this workload is depicted in Figure 5. The distribution is skewed towards left: most of the executed workflows explore fewer puzzle setups than the average of the whole workload. This indicates that in most cases the workflow termination condition (finding a puzzle setup that matches the difficulty settings) is met faster than the workload average indicates.

5.3 Meeting the Challenges

We have identified in Section 2.2 two main challenges for puzzle content generation, puzzle difficulty and scalability. We now show evidence that the POGGI architecture can meet these challenges.

We first evaluate the impact of the application configuration on finding puzzle instances of different difficulty. Two configurations are considered during this experiment: workflows that explore a normal-sized space (*normal instances*), and workflows that explore a large-sized space (*large instances*). Table 2 shows the number of puzzle instances found for these two configurations. The normal workflow instances find more puzzle instances with solution sizes up to 7. For solution sizes of 8 through 12, both instances behave similarly. For solution sizes of 13 and higher, the large workflow instances become the best choice. Based on similar results and on demand the Capacity Planning component can issue recommendations for efficiently finding unique instances of desired difficulty.

To show evidence of scalability we investigate the average response time and the potential for soft performance guarantees. For the 10,000 workflows test workload described in Section 5.2, the average workflow is computed in around 5 minutes; thus, it is possible to generate content for hundreds of thousands of players on a moderately sized grid infrastructure.

We have also evaluated the application- and the service-level throughput over time. We define the application-level (service-level) throughput as the number of (interesting) puzzle setups explored (found) over the time unit, set here to one second; for convenience, we use the terms states and puzzle setups interchangeably. Figure 6 shows the evolution of application- and service-level throughput over time. The architecture achieves an overall application-level throughput of over 15 million states explored per second, and an overall service-level throughput of over 0.5 interesting states discovered per second. The performance decreases with time

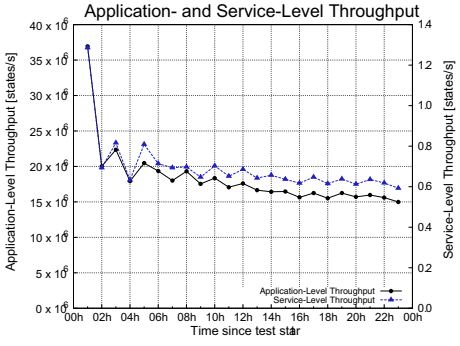

Fig. 6. The evolution of the application- and service-level throughput over time

due to Condor's fair sharing policy: our normal user's priority degrades with the increase of resource consumption. The performance decrease becomes predictable after about 6 hours. This allows a service provider to practically guarantee service levels, even in a shared environment. These experiments also demonstrate the ability of our architecture to extract application- and service-level metrics.

6 Related Work

In this section we survey two areas related to our work: game content generation, and many-tasks applications. We have already presented in Section 2.1 the current industry practice.

The topic of automated (procedural) generation has already been approached by the industry as an alternative to manual content generation. In 1984, the single-player game Elite [6] used automated game content generation to simulate a large world on an 8-bit architecture. Since then, several other games have used a similar approach: ADOM [7] generates battle scenes that adapt dynamically to the player level, Diablo [8] generates instances of enclosed areas for the player to explore, The Dwarf Fortress [9] generates an entire world from scratch, etc. All these approaches were games with a small numbers of concurrent players or even with a single player, and generated content on the (main) player's machine. In contrast, this work focuses on the efficient generation of puzzle game instances for MMOGs, using a large resource pool as the computing infrastructure.

Until a few years ago, few environments existed that could manage the high number of jobs required by MMOGs, among them SETI@Home [35]. More recently, tools such as Falkon [28] and Swift (through Falkon) have started to address the problem of executing with low overhead large numbers of bags-of-tasks and workflows, respectively. In contrast with all these approaches, our architecture optimizes the execution of a specific application (though with a much wider audience) and specifically considers dynamic resource provisioning adjustments to maintain the performance metrics required by application's commercial focus.

7 Conclusion and Ongoing Work

With a large and growing user base that generates large revenues but also raises numerous technological problems, MMOGs have recently started to attract the interest of the research community. In this work we are the first to identify the problem of the scalability of content generation. To address this problem, we have designed an implemented POGGI, an architecture for automatic and dynamic content generation for MMOGs. Our architecture focuses on puzzle game content generation, which is one of the most important components of the generic game content generation problem. Experimental results in a large resource pool show that our approach can achieve and even exceed the manual generation of commercial content.

Currently, we are extending our architecture with more game content types and with mechanisms for malleable workflow execution. For the future, we plan make POGGI less domain-specific, towards generic scientific computing support.

References

1. Woodcock, B.S.: An analysis of mmog subscription growth. Online Report (2009), http://www.mmogchart.com
2. Nae, V., Iosup, A., Podlipnig, S., Prodan, R., Epema, D.H.J., Fahringer, T.: Efficient management of data center resources for massively multiplayer online games. In: ACM/IEEE SuperComputing (2008)
3. Bartle, R.: Designing Virtual Worlds. New Riders Games (2003) ISBN 0131018167
4. Kasparov, G.: Chess Puzzles Book. Everyman Chess (2001)
5. Polgar, L.: Chess: 5334 Problems, Combinations and Games. Leventhal (2006)
6. Braben, D., Bell, I.: "Elite", Acornsoft, 1984 (2009), http://www.iancgbell.clara.net/elite/
7. Biskup, T.: "ADOM", Free (1994), http://www.adom.de/ (2009)
8. Schaefer, E., et al.: "Diablo I", Blizzard Entertainment (1997), http://www.blizzard.com/us/diablo/ (2009)
9. Adams, T.: "Dwarf Fortress", Free (2006), http://www.bay12games.com/dwarves/ (2009)
10. Neumann, C., Prigent, N., Varvello, M., Suh, K.: Challenges in peer-to-peer gaming. Computer Communication Review 37(1), 79–82 (2007)
11. White, W.M., Koch, C., Gupta, N., Gehrke, J., Demers, A.J.: Database research opportunities in computer games. SIGMOD Record 36(3), 7–13 (2007)
12. White, W.M., Demers, A.J., Koch, C., Gehrke, J., Rajagopalan, R.: Scaling games to epic proportion. In: ACM SIGMOD ICMD, pp. 31–42. ACM, New York (2007)
13. Müller, J., Gorlatch, S.: Rokkatan: scaling an rts game design to the massively multiplayer realm. Computers in Entertainment 4(3) (2006)
14. The Entertainment Software Association, "2008 annual report," Technical Report, http://www.theesa.com (November 2008)
15. Fritsch, T., Ritter, H., Schiller, J.H.: The effect of latency and network limitations on mmorpgs: a field study of everquest2. In: NETGAMES. ACM, New York (2005)
16. Iosup, A., Dumitrescu, C., Epema, D.H.J., Li, H., Wolters, L.: How are real grids used? the analysis of four grid traces and its implications. In: GRID, pp. 262–269. IEEE, Los Alamitos (2006)

17. Lublin, U., Feitelson, D.G.: The workload on parallel supercomputers: modeling the characteristics of rigid jobs. J. PDC 63(11), 1105–1122 (2003)
18. Iosup, A., Epema, D.H.J., Franke, C., Papaspyrou, A., Schley, L., Song, B., Yahyapour, R.: On grid performance evaluation using synthetic workloads. In: Frachtenberg, E., Schwiegelshohn, U. (eds.) JSSPP 2006. LNCS, vol. 4376, pp. 232–255. Springer, Heidelberg (2007)
19. Feitelson, D.G., Rudolph, L.: Metrics and benchmarking for parallel job scheduling. In: Feitelson, D.G., Rudolph, L. (eds.) IPPS-WS 1998, SPDP-WS 1998, and JSSPP 1998. LNCS, vol. 1459, pp. 1–24. Springer, Heidelberg (1998)
20. Raicu, I., Zhang, Z., Wilde, M., Foster, I., Beckman, P., Iskra, K., Clifford, B.: Toward loosely-coupled programming on petascale systems. In: ACM/IEEE SuperComputing (2008)
21. Iosup, A., Sonmez, O.O., Anoep, S., Epema, D.H.J.: The performance of bags-of-tasks in large-scale distributed systems. In: HPDC, pp. 97–108 (2008)
22. Conway, J.H.: All games bright and beautiful. The American Mathematical Monthly 84(6), 417–434 (1977)
23. Bouzy, B., Cazenave, T.: Computer go: An ai oriented survey. Artificial Intelligence 132, 39–103 (2001)
24. Demaine, E.D.: Playing games with algorithms: Algorithmic combinatorial game theory. In: Sgall, J., Pultr, A., Kolman, P. (eds.) MFCS 2001. LNCS, vol. 2136, pp. 18–32. Springer, Heidelberg (2001)
25. Yamamoto, H., Yoshigahara, N., Tanaka, G., Uematsu, M., Nelson, H.: Lunar Lockout: a space adventure puzzle, ThinkFun (1989)
26. Stratan, C., Iosup, A., Epema, D.H.J.: A performance study of grid workflow engines. In: GRID, pp. 25–32. IEEE, Los Alamitos (2008)
27. Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: the condor experience. Conc.&Comp.: Pract.&Exp. 17, 323–356 (2005)
28. Raicu, I., Zhao, Y., Dumitrescu, C., Foster, I.T., Wilde, M.: Falkon: a fast and light-weight task execution framework. In: ACM/IEEE SuperComputing (2007)
29. Singh, G., Kesselman, C., Deelman, E.: Optimizing grid-based workflow execution. J. Grid Comput. 3(3-4), 201–219 (2005)
30. Ludäscher, B., et al.: Scientific workflow management and the Kepler system. Conc. & Comp.: Pract. & Exp. 18(10), 1039–1065 (2006)
31. Oinn, T.M., et al.: Taverna: lessons in creating a workflow environment for the life sciences. Conc. & Comp.: Pract. & Exp. 18(10), 1067–1100 (2006)
32. von Laszewski, G., Hategan, M.: Workflow concepts of the Java CoG Kit. J. Grid Comput. 3(3-4), 239–258 (2005)
33. Truong, H.L., Dustdar, S., Fahringer, T.: Performance metrics and ontologies for grid workflows. Future Gen. Comp. Syst. 23(6), 760–772 (2007)
34. Iosup, A., Epema, D.H.J.: GrenchMark: A framework for analyzing, testing, and comparing grids. In: CCGrid, pp. 313–320. IEEE, Los Alamitos (2006)
35. Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M., Werthimer, D.: "SETI@Home". Commun. ACM 45(11), 56–61 (2002)