Distributed Systems Group
Department Software Technology
Faculty EEMCS
TU DELFT

LARGE LAB EXERCISE B

DISTRIBUTED COMPUTING SYSTEMS (IN4391)

The Dragons Arena System:
Distributed Simulation of Virtual Worlds

J.S. Rellermeyer

v.1.0, February 13, 2018

A Learning Objectives

1. Implement complex operations of cloud computing in realistic scenarios.
2. Analyze the tradeoffs inherent in the design of cloud-computing-based applications.

This exercise combines elements of Chapter 4 (communication), Chapter 6 (synchronization),
Chapter 7 (consistency and replication), and Chapter 8 (fault tolerance) of the course text book
(Andrew S. Tanenbaum and Maarten van Steen, *Distributed Systems: Principles and Paradigms*, second edition, Prentice Hall, 2007).

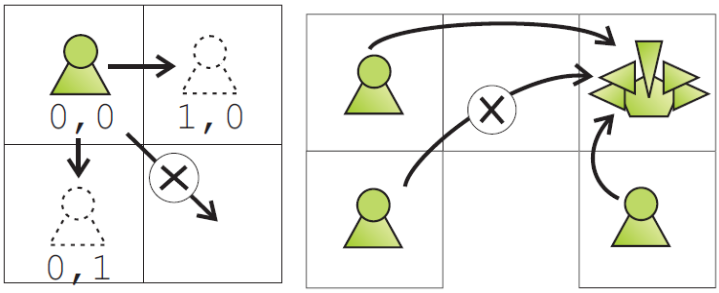

Figure 1. Rules of Dragons Arena:
(left) a player can move horizontally or vertically, but not both at once;
(right) a player can hit a dragon (or vice versa) from a distance of at most 2 squares.

B Assignment

Computer games are increasingly present in our lives. Every day, hundreds of millions of players around the world are entertained by games such as FarmVille, World of Warcraft, The Sims, and DotA 2. The US Entertainment Software Association (ESA) reports that, in 2012, over two thirds of the American households play computer or video games; inside these households, the average game player age is over 35, and the average experience with computer or video games is 12 years. In a market that now exceeds $30bn./year, start-up companies cannot afford investing in R&D for both content and technology. However, supporting massive amounts of concurrent users requires building a distributed system, with many unknowns regarding performance, scalability, consistency, etc. Thus, a market opportunity exists for consultants who can help start-up gaming companies to setup their infrastructure.

Your team ("you") is hired by WantGame BV to develop their new game engine. Because the computational complexity of these engines is high, and because the CTO of WantGame BV believes distributed systems offer an excellent performance-cost trade-off when supporting many concurrent users, WantGame BV has decided to invest in the design and implementation of a distributed game engine, the Dragon Arena system (the DAS[1]). **You must design, develop, experiment with realistic scenarios, and report back on the feasibility of a distributed DAS.**

With their DAS, WantGame BV intends to run a service of online warfare between computer-controlled dragons and hundreds of virtual knights (avatars or real-life humans). Their game design is rather simplistic:

1. The virtual world is a grid battlefield of size of 25x25 squares.
2. Each battle participant, be it a player or a dragon, occupies a single square, and different battle participants cannot occupy the same square.
3. Each battle participant maintains its health status (its initial and remaining number of health points (`hp`)), and has a certain strike power (attack points, `ap`). Initially, each player has 10-20 `hp` and 1-10 `ap`, and each dragon has 50-100 `hp`

[1] Do not confuse DAS, the distributed gaming engine, with DAS-5, the real multi-cluster system on which you can perform experiments. DAS-5 is introduced in Section G.

and 5-20 `ap`.

4. Players and dragons may perform actions, as follows. Players can move (see Figure 1a) on the battlefield 1 step horizontally or vertically, but not both during the same action; dragons cannot move. Players can strike closely located dragons and vice versa. The effect of striking is the subtraction of the attacker's `ap` from the victim's remaining `hp`, and closely located means from a distance of at most 2 (see Figure 1b; the distance between two squares is the sum of the horizontal and vertical distance between them). When a participant's remaining `hp` drops to 0 or below, the participant is removed from the battlefield. Players can heal nearby players, with the effect of adding to the receiver's remaining `hp` an amount equal to the healer's `ap`, up to the receiver's maximum (initial) `hp`; here, nearby means from a distance of at most 5.
5. There must be a certain delay between two consecutive actions of the same battle participant.

A non-distributed version of a DAS (see Figure 1) has been implemented by a previous consultant of WantGame BV and is available for study. You may also want to derive inspiration for this exercise from the paper (available from the Lab site on Blackboard):

> Eric Cronin, Burton Filstrup, Anthony R. Kurc, and Sugih Jamin, *An efficient synchronization mechanism for mirrored game architectures*, NETGAMES, 2002.

(Mandatory requirements) You must design and implement a distributed (i.e., with multiple server nodes), replicated, and fault-tolerant version of the DAS, and assess its properties. The DAS system should meet the following requirements:

1. *System operation requirements:* You must implement the game design specified by WantGame BV. The actions of players—connecting to and disconnecting from the DAS system, and taking in-game actions—need to be emulated, that is, must be replaced with computer-controlled commands that emulate human behavior. For simplicity, assume that the players' in-game actions are derived from a simple strategy: heal a nearby player as soon as there is one that has below 50% of its initial `hp`, and go towards the closest dragon and strike otherwise. **For connecting to and disconnecting from the system, WantGame BV wants to use data from a real workload trace taken from the Game Trace Archive [5].**

2. *Fault tolerance:* Consider a simple failure model, in which client and server nodes may crash and restart. The DAS must by design be resilient against client and server node crashes. In addition, all game and system events (e.g., player moves, strikes, client and server node restarts) must be logged in the order in which they occur, on at least two server nodes.

3. *Scalability requirements:* The properties of the DAS must be demonstrated when it contains 100 players, 20 dragons, and 5 server nodes, and when it runs until there is only one class of remaining participants, e.g., either players or dragons.

To ascertain the correct completion of the exercise you have to write a report of 4-6

pages, to be filled as described in the following, and to be assessed in three stages.

(Optional, for bonus points, see Section F) Additional features that you may want to consider for your system (pick at most two):

1. *Multiple consistency models*: analysis of the impact of the consistency model on performance.
2. *Advanced fault-tolerance:* through tolerance for and analysis of the impact of multiple failures or other failure models than fail-stop on the DAS system.
3. *Multi-tenancy*: support multiple users running simulations for the DAS system, through a queue of simulation jobs and adequate scheduling policies.
4. *Repeatability*: making sure that the outcome of the simulations is always the same for the same input, through the use of priority numbers for concurrent simulated events [1, Section 3.9.2].
5. *Benchmarking*: by running service workloads designed to stress particular elements of the system.
6. *Security*: by implementing access lists, security groups, and a byzantine fault-tolerance algorithm, etc.
7. *Portfolio scheduling*: by adapting the concept of portfolio scheduling to the Dragon Arena System.

Good guidelines for this work can be found in previous publications of the PDS group [2,3,4].

C Deadlines

1. *(recommended) February 28, 2018 (week 3.3):* Discuss with TA your group's system requirements and proposed approach for replication and consistency, for this lab exercise.
2. *(recommended) March 14, 2018 (week 3.5):* Discuss with TA your group's system requirements and proposed approach for fault-tolerance and scalability, for this lab exercise.
3. **(mandatory) anytime before March 28, 2018 (week 3.7): Demonstrate to TA the system implemented for this exercise.**
4. **(mandatory) March 28, 2018 (week 3.7): Turn in to TA the report for this exercise.**

To achieve the above-mentioned deadlines, your assignment can be divided into three parts:

Weeks 2—3:

- Analyze the system functionality and summarize its requirements.
- Analyze the potential system implementation and summarize your design's key features. Discuss with your team-mate the alternatives for the design choices you have made, and summarize them in your report. Try to have the resulting requirements and design document (maximum 2 pages) approved by the assistants before you start the actual implementation.

- Implement *replication* and a suitable *consistency* mechanism.

Weeks 4—5:

- Implement *fault-tolerance*.
- Test your system's implementation for *scalability* (see the scalability requirements, point 3 in the mandatory requirements).
- Consider appropriate failure rates and recovery durations. Make **a test plan for scenarios with failures (maximum 0.5 pages)**, and try to have it approved by the assistants.

Weeks 6—7:

- Implement and execute the test plan with failures.
- Demonstrate your system to the assistants.
- Finish your report by including the achieved results and your conclusions, and have it approved by the assistants.

Note: the estimated time required for the completion of WantGame's assignment (the large exercise) is 112 hours, equally divided by a team of two.

Notes:

- Try not to exceed, for any experiment, 10 hours of work.
- You can leverage the same development and setup for several experiments. In this case, report the time spent for the shared part only for the first experiment and explain the large amount of time spent for it in the report.

D Report Structure

The report should have the following structure:

1. *Report title, authors, and support cast* (Lab assistant and course instructors). For each person in your team, give name and contact information (email).
2. *Abstract*: a description of the problem, system description, analysis overview, and one main result. Size: one paragraph with at most 150 words.
3. *Introduction (recommended size, including points 1 and 2: 1 page)*: describe the problem, the existing systems and/or tools about which you know (related work), the system you are about to implement, and the structure of the remainder of the article. Use one short paragraph for each.
4. *Background on Application (recommended size: 0.5 pages)*: describe the DAS application (1 paragraph) and its requirements (1 paragraph per each of consistency, scalability, fault-tolerance, and performance).
5. *System Design (recommended size: 1.5 pages)*
    a. *System overview*: describe the design of your system, including the system operation, fault tolerance, and scalability components (which correspond to the homonym features required by the WantGame CTO).
    b. **(Optional, for bonus points, see Section F)** *Additional System Features*: describe each additional feature of your system, one sub-section per feature.

6. *Experimental Results (recommended size: 1.5 pages)*
    a. *Experimental setup*: describe the working environments (DAS-5, Amazon EC2, etc.), the general workload and monitoring tools and libraries, other tools and libraries you have used to implement and deploy your system, other tools and libraries used to conduct your experiments.
    b. *Experiments*: describe the experiments you have conducted to analyze each system feature, such as consistency, scalability, fault-tolerance, and performance. Analyze the results obtained for each system feature. Use one sub-section per experiment (or feature). In the analysis, also report:
        i. *Service metrics of the experiment*, such as runtime and response time of the service, etc.
        ii. *(optional) Usage metrics and costs of the experiment*.

7. *Discussion (recommended size: 1 page)*: summarize the main findings of your work and discuss the tradeoffs inherent in the design of the DAS system. Should WantGame use a distributed system to implement the DAS system? Try to extrapolate from the results reported in Section 6.b for system workloads that are orders of magnitude higher than what you have tried in real-world experiments.
8. *Conclusion*
9. *Appendix A: Time sheets (see Section E)*

E Document the Time You Spend

Because you conduct a consultancy job for WantGame, you should report on the time it takes to conduct each major part of the assignment. Specifically, report:

- the `total-time` = total amount of time spent in completing the assignment (the large exercise).
- the `think-time` = total amount of time spent in thinking about how to solve the assignment (the large exercise).
- the `dev-time` = total amount of time spent in developing the code needed to solve the assignment (the large exercise).
- the `xp-time` = total amount of time spent in experiments for the assignment (the large exercise).
- the `analysis-time` = total amount of time spent in analyzing the results of the experiments for the assignment (the large exercise).
- the `write-time` = total amount of time spent in writing this report
- the `wasted-time` = total amount of time spent in activities related to the assignment (the large exercise), but which cannot be charged as think-time, dev-time, xp-time, analysis-time, or write-time.

F Scoring

1. **Quality of the report [8,9]** (presentation, style of writing, graphing, requirements analysis, design, analysis of results): **+2,000 points**.
2. **Technical quality of the systems work** (basic system features are supported)**: +2,000 points**.

3. *Bonuses*
    a. Making your source code open-source and your report public: +100 points.
    b. Additional system features (feature and report): +250 points/feature.
    c. Excellent report (writing, graphing, description of system): at most +500 points.
    d. Excellent analysis (design of experiments, analysis of results): at most +500 points.

4. *Limits on bonuses*
    a. **The total additional points for this exercise can amount to at most +2,000 points, representing additional features, excellent report and/or analysis bonuses, etc.**
    b. The additional features (see page 3) can amount to at most +1000 points. In other words, you can receive bonuses for at most 4 well-supported additional features.

Note: writing a good technical report is scored *equally* to writing code for a system that works! This means that you should schedule time to write a good report, and for one revision to respond to our feedback.

G Use of Existing Technology

You are not limited for this exercise to any technology, but:

- **The non-distributed version of the DAS is available, as a Java program, in the file** `IN4391-DAS.zip`. **This version is provided as-is, for study purposes only, without even implied guarantees that it will compile on your system.**

- You CAN use the machines provided by our system DAS-5, or by Amazon EC2 or another IaaS cloud. For Amazon EC2, you can use the free resources provided through the Free Usage Tier [http://aws.amazon.com/free/]; if you use the `m1.small` (paid) instances of Amazon EC2, the estimated cost for this exercise does not exceed 8 EUR[2].
- You MUST NOT use libraries that already provide the functionality required in Section B. Should you use such libraries, you MUST NOT use from them the parts that provide the functionality required in Section B. For example,
    - **you CAN use Java RMI.**
    - **you CAN use Python's default communication libraries (sockets).**

When using an external library, give due credit in your code and report (see also Section H).

When in doubt about technology you intend to use, consult with the lab assistant.

[2] At 8 Euro-cents per hour, assuming a total workload of 100 hours. Mileage may vary.

H Anti-Fraud Policy (Zero-Tolerance)

Our anti-fraud policy is simple: zero-tolerance, within the limits set by TU Delft. We will pursue each case of potential fraud we discover. We will use to the maximum extent the means provided by TU Delft to punish (attempts to) fraud.

You can learn more about how to prevent that you commit fraud via a discussion with the studieadviseur, from Appendix 2 in the Toetsbeleid TU Delft [6], or even from international sources such as Harvard's guidelines on avoiding plagiarism [7].

References

[1] Richard Fujimoto, Parallel and distributed simulation systems, John Wiley & Sons, 2000.

[2] Vlad Nae, Alexandru Iosup, Radu Prodan: Dynamic Resource Provisioning in Massively Multiplayer Online Games. IEEE Trans. Parallel Distrib. Syst. 22(3): 380-395 (2011). [Online] Available: http://doi.ieeecomputersociety.org/10.1109/TPDS.2010.82 or http://www.st.ewi.tudelft.nl/~iosup/mmog-data-centers10tpds.pdf .

[3] Siqi Shen, Otto Visser, Alexandru Iosup: RTSenv: An experimental environment for real-time strategy games. NETGAMES 2011: 1-6. [Online] Available: http://www.pds.ewi.tudelft.nl/~iosup/PDS-2011-002.pdf (extended version).

[4] Y. Guo, S. Shen, O. Visser, and A. Iosup, An Analysis of Online Match-Based Games, In International Workshop on Massively Multiuser Virtual Environments (MMVE 2012), October 08-09, 2012, Munich, Germany; held in conjunction with the IEEE International Symposium on Audio-Visual Environments and Games (HAVE 2012). [Online] Available: http://www.st.ewi.tudelft.nl/~iosup/match-based-games12mmve.pdf .

[5] Yong Guo, Alexandru Iosup: The Game Trace Archive. NetGames 2012: 1-6. [Online] Available: http://dx.doi.org/10.1109/NetGames.2012.6404027 and http://www.pds.ewi.tudelft.nl/~iosup/game-trace-archive12netgames.pdf .

[6] Kefeng Deng, Junqiang Song, Kaijun Ren, Alexandru Iosup: Exploring portfolio scheduling for long-term execution of scientific workloads in IaaS clouds. SC 2013: 55 [Online] Available: http://doi.acm.org/10.1145/2503210.2503244 and http://www.pds.ewi.tudelft.nl/~iosup/portfolio-scheduling13sc.pdf .

References for Anti-Fraud Policy

[7] TU Delft's anti-fraud policy, Appendix 2 in Toetsbeleid TU Delft, 2010. [Online] Available: https://intranet.tudelft.nl/live/pagina.jsp?id=4db657d4-36f3-49b0-9666-97ddea61139f&lang=nl&binary=/doc/3.%20Toetsbeleid%20TUD%20(12-11-2010)%20def.pdf (last accessed Sep 2012).

[8] Avoiding Plagiarism, Harvard web site document, 2012. [Online] Available: http://isites.harvard.edu/icb/icb.do?keyword=k70847&tabgroupid=icb.tabgroup106849 (last accessed Sep 2012).

References for Quality of Writing

[9] Strunk and White, The Elements of Style. Longman, Fourth Edition, 1999.

[10] Simon Peyton Jones, How to write a good research paper. [Online] Available: http://research.microsoft.com/en-us/um/people/simonpj/papers/giving-a-talk/giving-a-talk.htm . Last retrieved: 10-Feb-2014.