PSBench: A Benchmark for Content- and Topic-based Publish/Subscribe Systems

Kaiwen Zhang
University of Toronto

Tilmann Rabl
University of Toronto

Yi Ping Sun
University of Toronto

Rushab Kumar
University of Toronto

Nayeem Zen
University of Toronto

Hans-Arno Jacobsen
University of Toronto

ABSTRACT

The publish/subscribe paradigm has found wide acceptance in a broad variety of use cases that differ dramatically in the characteristics of their workloads. Many different systems have been developed both by academia as well as industry, but there is no definitive benchmark, which enables a fair comparison between the different systems.

In this demo, we present PSBench, a benchmark specification and suite for publish/subscribe systems that covers a broad variety of publish/subscribe workloads and scenarios. The benchmark suite is extensible and generic, but the specification targets social games. Social games are the ideal use case since they have a very broad range of requirements and produce a variety of publications and subscriptions. We draw from our experience in massive multi-player online games to construct a highly realistic workload. In this demo, we present the toolchain, the workload and the graphical interfaces that enable an extensive performance evaluation of publish/subscribe systems.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering; H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

Keywords

Publish/subscribe, content-based matching, topic-based matching, benchmarking

1. INTRODUCTION

There is a healthy ecosystem of publish/subscribe systems built by the academia and industry. However, there is currently no commonly agreed upon benchmark for pub/sub: every publication uses its own workload showcasing their system's sweet spot. The lack of benchmarks also results in a lack of standardization in general, since standard benchmarks result in a common denominator and common interfaces, as observed in other benchmarking efforts such as database benchmarking.

Middleware '14, December 08 - 12 2014, Bordeaux, France
Copyright 2014 ACM 978-1-4503-3220-0/14/12$15.00
http://dx.doi.org/10.1145/2678508.2678517

Benchmarks can be categorized into different models. Component-level benchmarks test certain components of the system under test (such as the matching engine of a pub/sub system), while application level benchmarks try to capture all performance aspects of a certain application. The workloads can be purely synthetic, realistic, or real. For component-level benchmarks, purely synthetic workloads are useful to test a component in specific situations. Application-level benchmarks typically use realistic data, that covers a range of situations that a real use case would exhibit. While real data sets are good to test for the application that they are take from, they are not scalable and often not generalizable.

In this demo, we present PSBench, an application-level benchmark for publish/subscribe systems. While its tool chain is built as a generic framework, it features an application-level scenario based on social games. In our demo, we will show all the steps to benchmark a publish/subscribe system such as PADRES [4].

2. APPLICATION SCENARIO

Typical pub/sub applications include stock trading [8], supply chain management [7], and Twitter-like applications. Content-based matching is usually required for location-based services, e.g., smart traffic management [5]. We chose social games as application scenario, since social games have been shown to be benefit from employing publish/subscribe systems [1, 3] and the scenario covers a broad variety of workloads.

Within the social game scenario, there are several sub-cases for pub/sub systems. A content-based scenario is state update dissemination. In massive multiplayer online games, typical publications are location updates, object updates, players joining and leaving the game. Subscriptions cover ranges around the players position. Another use case is chatting within the game: this is a topic-based scenario, where there can be private chats between two players, chatrooms, or broadcast announcements. An important source of revenue in social games are advertisements and in-game item selling. In this scenario, the users' interests specify the subscriptions and item listings are the publications. Finally, metrics and statistics can be gathered using pub/sub systems. These scenarios can be used separately or combined.

3. ARCHITECTURE

The benchmark suite is comprised of a generic workload driver and a use case dependent workload generator. A high level overview

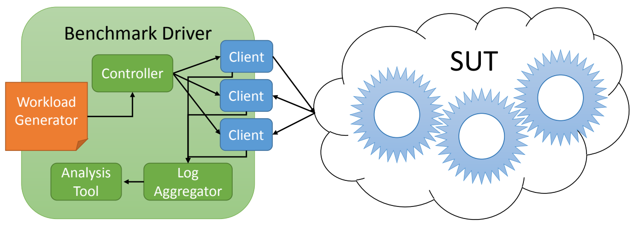


Figure 1: PSBench Architecture

can be seen in Figure 1. The system under test (SUT) is considered a black box. In order to support as many systems as possible there is a special client interface that has to be implemented for each system to be tested, since there is no standard interface for pub/sub systems. The core of the benchmark driver is the controller that starts the correct amount of clients and manages the feeding of the workload from the workload generator. The workload generator specifies the workload according to the targeted use case. The log aggregator collects all metrics collected by the clients and provides them to the analysis tool, that presents the measured performance in an easy readable manner.

4. WORKLOAD GENERATOR

As mentioned above, the workload generator is a plug-in to the benchmark driver. It generates workloads in form of time series of advertisements, publications, subscriptions, and their negations. The interface is largely consistent with the API presented in [6]. The social game workload generator is mostly inspired by the massive multiplayer online game Mammoth, developed at McGill University [5].

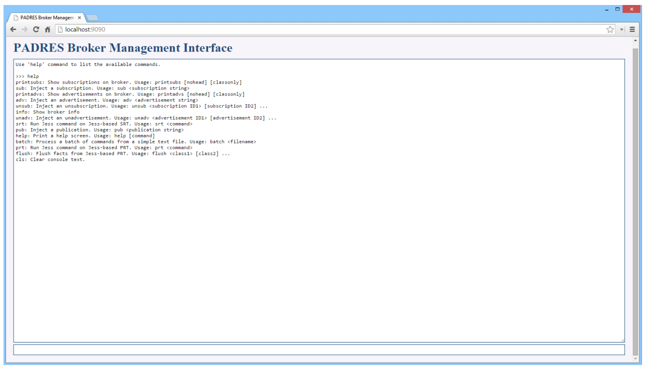
The workload generated is stored in separate log files, which are to be provided to each client during the experiment. Below is a sample of our content-based position update scenario:

```
100 p [class,'pos'],[client,1],[x,4.51],[y,9.20]
100 s [class,eq,'pos'],[client,eq,1],[x,in,1.51,7.51],
      [y,in,1.51,12.20]
100 p [class,eq,'pos'],[client,eq,1],[x,2.51],[y,7.20]
100 us 1
```

Each line is a separate publish/subscribe command to be submitted from the client parsing this log file. The first number at the beginning of each line is the delay in milliseconds from the previous command. This is followed by the type of command (eg. p for publish, s for subscribe, etc.). Finally, the rest of the line is the parameters of the command. For example, a publication command contains all the attribute-value pairs of the publication. This sample shows the workload of a player moving a map, updating its position and subscribing to the radius around its current location.

5. BENCHMARK DRIVER

Our benchmark driver deploys and orchestrates a distributed benchmarking experiment according to the parameters provided. In particular, the driver distributes a separate workload log file to each client involved in the experiment. The log file is parsed during the experiment to pass publish/subscribe commands (such as sending a subscription) at specific timestamps to the underlying pub/sub system. Each system tested must implement a bridge API to process the commands internally.



PADRES Broker Management Interface

```
Use 'help' command to list the available commands.

>>> help
printsubs: Show subscriptions on broker. Usage: printsubs [nohead] [classonly]
sub: Inject a subscription. Usage: sub <subscription string>
printadvs: Show advertisements on broker. Usage: printadvs [nohead] [classonly]
adv: Inject an advertisement. Usage: adv <advertisement string>
unsub: Inject an unsubscription. Usage: unsub <subscription ID1> [subscription ID2] ...
info: Show broker info
unadv: Inject an unadvertisement. Usage: unadv <advertisement ID1> [advertisement ID2] ...
srt: Run Jess command on Jess-based SRT. Usage: srt <command>
pub: Inject a publication. Usage: pub <publication string>
help: Print a help screen. Usage: help [command]
batch: Process a batch of commands from a simple text file. Usage: batch <filename>
prt: Run Jess command on Jess-based PRT. Usage: prt <command>
flush: Flush facts from Jess-based PRT. Usage: flush <class1> [class2] ...
cls: Clear console text.
```

Figure 2: Web Monitor

6. LIVE DEMO

Our live demo will mainly focus on benchmarking PADRES [4]. We will show how to generate a workload using our generator and how to adjust parameters such as the number of players, their distribution in the map, the frequency of movement, etc. We then show to deploy the newly generated log files into our driver and observe the results using our monitor tool (see Figure 2). Other pub/sub systems, such as SIENA [2], will be available for testing.

7. REFERENCES

[1] C. Cañas, K. Zhang, B. Kemme, J. Kienzle, and H.-A. Jacobsen. Publish/Subscribe Network Designs for Multiplayer Games. In *Middleware*, 2014.

[2] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Trans. Comput. Syst.*, pages 332–383, 2001.

[3] M. Fan and R. Chen. Real-time Analytics Streaming System at Zynga. In *XLDB*, 2012.

[4] H. Jacobsen, A. K. Y. Cheung, G. Li, B. Maniymaran, V. Muthusamy, and R. S. Kazemzadeh. The PADRES Publish/Subscribe System. In *Principles and Applications of Distributed Event-Based Systems*. IGI Global, 2010.

[5] N. K. Pandey, K. Zhang, S. Weiss, H.-A. Jacobsen, and R. Vitenberg. Distributed Event Aggregation for Content-based Publish/Subscribe Systems. In *DEBS*, 2014.

[6] P. Pietzuch, D. Eyers, S. Kounev, and B. Shand. Towards a Common API for Publish/Subscribe. In *Proceedings of the 2007 Inaugural International Conference on Distributed Event-Based Systems*, DEBS '07, pages 152–157, New York, NY, USA, 2007. ACM.

[7] K. Sachs, S. Kounev, J. Bacon, and A. Buchmann. Workload Characterization of the SPECjms2007 Benchmark. In *EPEW*, 2007.

[8] M. Sadoghi, M. Labrecque, H. P. Singh, W. Shum, and H.-A. Jacobsen. Efficient Event Processing through Reconfigurable Hardware for Algorithmic Trading. In *VLDB*, 2010.