TP3 Individual Writing Exercise: Performance monitoring using WebSockets

by Maksim Solovjov, 1104699

Preliminaries

Here are the necessary terms that a reader may not be familiar with. Technology-savvy readers might feel free to skip this section of the report.

Performance monitoring – observing the statistics of some system, which relate to how well does the system run; for example monitoring the CPU load, in order to ensure, that it does not exceed some healthy threshold.

Cloud-based solution – the program that is provided as a service on some external machines, which might be owned by a separate organisation, accessible via internet.

Client – in our case, a user-facing application, running in user's browser.

Server – a computation-intensive application, which serves client to the user and provides the gateway to the database.

Daemon – a monitoring application, running on the machine that is to be monitored, storing the performance metrics and sending them to the Server and Client.

Relational database – a database, modelled via creating entities and expressing their relations to each other, interaction with which is governed by relational algebra.

NoSQL database – an alternative to the relational database, which allows easy horizontal scaling (can accommodate more data), in our case (MongoDB) expressed via collections of documents with no enforced schema (blueprint).

WebSocket – a protocol, that allows simultaneous bidirectional communication over a single TCP connection. To put it in simple terms, it lets a user to both send and receive data in real-time without the overhead of making multiple "internet" connections.

Concurrency – splitting work into independent chunks, that can be done in no particular order, allowing switching from one task to another, should the first one temporarily stall.

Parallelism – executing multiple concurrent tasks at the same time.

Refactor – rewrite, in order to improve the quality of the code, without changing its output.

Motivation

As the computing power grows, so do our ambitions, which, in turn, require even more computing power to satisfy them. Quite often, this race forces companies to deploy whole datacentres of machines. This poses many problems, one particularly important being how to ensure the whole infrastructure runs smoothly.

The key step in any solution for this, be it automated or manual, is to monitor how is the system performing. A multitude of different parameters might be used to express the performance; however, for the most systems it is enough to know:

- CPU load,
- Network load,
- Memory usage,
- Temperature,
- Accessibility of the system.

This data should be available in real-time, since otherwise there is an artificial delay in how quickly can one respond to the emerging problem. Finally, the installation of such a system should be as easy as possible, since there are many other things that system administrators already have to setup, and our aim is to make their life easier.

Now, there is a multitude of different systems that satisfy at least some of these requirements:

ZABBIX 1.8: supports most of the modern operating systems, monitors all of the aforementioned parameters if you install its agent, provides agentless monitoring of certain standard services like server uptime, has a web-interface and one underlying server for it. However, it is not offered as a Cloud service, you have to run the server yourself.

Ganglia: very scalable and robust, mostly used to monitor big clusters. It has a somewhat outdated, however functional, web interface, featuring basic graphs and an overview of the systems health.

Paessler PRTG Network Monitor 8: monitors around 190 different parameters, so that you can see the exact state of your database or which USB is being used. Features easy installation, however it possesses complicated interface, as a result of being such a heavyweight solution.

Monitor.us: optimised for home and small business use, cloud based, user-friendly, however still extremely functional, offering lots of features to monitor. It does not allow real time access to the data, with free and paid plans offering different fixed refresh rates.

Most of these solutions are feature bloated and overly complex, and as a result, they provide complicated and cluttered interfaces for the end-users. The easiest to setup solutions tend to be cloud-based; unfortunately, these usually do not provide real-time access to metrics. All things considered, there is a niche for a better and simpler product.

One of the key aspects of our product is that it allows easy cloud-based access while still providing real-time metrics. This became possible due to an emerging technology of WebSockets[1], which we

[1] The RFC for WebSockets is available here: http://tools.ietf.org/html/rfc6455.

utilise heavily. Its recently improved support in all the major browsers served as an inspiration for this project.

Aims

The project was suggested by J.P.Morgan, who are interested in its outcomes. Considering the specifics of their business, we had to provide a safe and robust solution. Aside from that, we wanted to provide them with something they could not get elsewhere, and thus we have committed an extensive market research. As a result, we have devised that the aim of the project is to provide a simple cloud-based solution for the real-time monitoring of the performance of most of the standard machines. Combining these three characteristics was the only way to stand out from our other competitors.

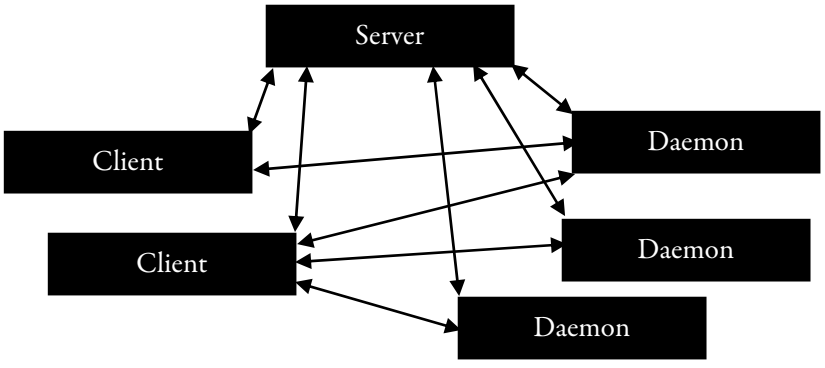
We had to be simple and usable, because we could not possibly compete in functionality. There are solutions out there that allow users to monitor almost every statistic imaginable; however, abundance of functionality takes its toll on the friendliness of the user interface. Simplicity makes difference, as it makes the work with the tool pleasant, almost seamless.

We wanted our project to be cloud-based (i.e. we provide it as a service, not as a product), because this reduces the installation overhead. We encourage our users to improve their operational excellence; hence, we have to provide an easy start both to the newcomers and seasoned administrators, switching to us from some another tool.

Finally, we want to provide a real-time solution, in order to decrease our customers' response time to any issues that might arise.

Progress

At the beginning of the semester, we have conducted a market research, decided upon the characteristics and architecture of the project and chose the right tools to implement the system. We have established that to meet our goals the project will be done as an ecosystem of many Clients, many Daemons and one cloud-based Server. These semi-independent parts communicate to each other using WebSockets and adhering to the rules, predefined by us.



Schematic representation of our Architecture

We chose client to be fully browser-based, written in CoffeeScript, using Backbone+Marionette combination as a framework. In this way, we could provide the best user interface with the minimum effort. It was decided, that we will write server in Go, since it made it very easy to write concurrent code, which could later be parallelised, hence an amazing fit for the cloud. We chose C++ to write Daemons, because they had to be very performant, leaving minimum trace in the system. They also had to have an access to the low-level system information. We decided to use Boost library for C++ to improve our productivity.

Afterwards we dived into coding and produced first working prototype soon afterwards, showing the viability of the technology chosen and that every part of the project can communicate to each other. The prototype could wire the commands through from client to daemon and vice-versa, as well as report the CPU usage to the client directly from the daemon in real-time.

The next step was to refactor the prototype. We have devised a way to perform automatic testing on a server easier, which allowed increasing its test coverage. The Daemon was restructured to make it easier to port to other platforms. The client was properly rewritten using the Backbone framework, so that the code is organised cleanly.

Afterwards we proceeded to implement a demo version of the product. Daemons could now report Network and Memory usage. Creation of the connection between the Daemon and the Client was supervised by server, to make it safe. Client was able to support more connections to more than one daemon. Overall, the product solidified and was ready to the streamlined development.

This is roughly the state of the project at the moment. We are planning to develop a first fully working version of the product soon, and spend the rest of the time improving it.