University of Helsinki
Antti Myyrä
014012055

Distributed Systems Project

1. Goals and methods

Goal of the course was the creation of distributed architecture in practice. In the assignment we needed to create different multitier architectures where calculations where partly or completely sent to the backend server for processing. In different steps the server received anything from the complete sin(x) plot for graph creation to just the basic calculations *(+, -, *, /)*.

Tools available in the front-end were limited to jQuery *(version 1.12.4)* and use of standard math library was forbidden, as sin(x) was meant to be calculated by using only the basic calculations that were in the last implementations sent to the server. This meant creating your own sin(x) function and helper functions for its calculation.

I chose the Taylor series as my form of sin(x) estimation. This required additional functions for power and factorial calculations, but worked rather well: error rate of less than 1% was achieved with just 8 iterations. Amount of calculations needed was still remarkable as creation of a sin(x) plot from -π to π required 11151 calculations.

2. Steps and their implementation

I chose javascrip as my backend language due to an earlier experience with Node.js and the possibility to share code with the frontend if needed. Complete server implementation was quite short, less than 150 lines of code.

2.1. Step 1: A distributed calculator

After the server was written, creating the first step was pretty straightforward. Splitting of the calculation into smaller parts with splitArguments function took the most time, but other parts of the frontend were easily created.

2.2.1. Step 2, variation 1: Gnuplot as a Service

As n*sin(x) was calculated in the client and then rendered on server with Gnuplot, the server needed an extra feature for this. Changes to frontend required the creation of the sin(x) plotpoints with Taylor series and the necessary functions for it, starting from power and factorial that can usually be found from the standard math library, but were now forbidden. With these done, rest was relatively easy.

In the backend, gnuplot image creation was quite hard as the datapoints were provided with HTTP GET and weren't in a correct order from the start. After processing them to correct form, Gnuplot needed to have them on disk for operating on them. Gnuplot's output was only saved on disk and needed to be retrieved for response. Other problem was caused by Node's gnuplot npm module that didn't have a completion callback, needing a separate timeout while we waited for it to finish processing.

2.2.2. Step 2, variation 2: Local sin(x) calculator

Changes in variation 2 were completely on frontend, as we didn't create any server queries. To

simplify things, I removed the support for any other calculations than n*sin(x), so possibilities like 2+3-1*sin(x) won't work with this variation (orders do support it). Most challenging thing in this variation was the use of HTML5 canvas element, that I had never used before. After a few examples and failed attempts, I managed to calculate the correct x and y on the canvas from the x and y in the plot. After this, adding X and Y axises and their legends was relatively easy.

2.2.3. Step 2, variation 3: Remote sin(x) calculator

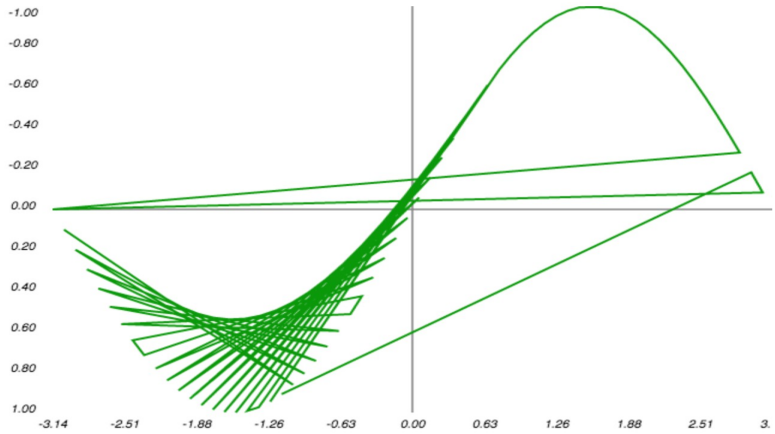
In this variation everything was calculated on the server and sin(x) plot created on clientside. Plot creation was already done, so it didn't need any changes, but the whole sin(x) calculation had to be rewritten. As every calculation was done on the server, Javascript's asynchronous nature required that each calculation also had a callback to to use when the calculation was completed. Power and factorial functions also had to support their own callbacks as they called themselves recursively.

After completion the calculator worked, but sheer number of calculations sent to serverside *(whole 11151 of them)* caused a delay not acceptable for web applications.

2.3. Step 3: Caching of sin(x) calculations and a simplify button

Last part of the project was the implementation of request cache and then caching the calculations processed in the server. After creating the cache, I changed the queryServer function to check cache before making a request. This was the best place to do it as all sin(x) building functions already called queryServer for calculations.

When experimenting with the cache, I ran into problems with cache updates as it gave erroneous answers when the same items was both being updated and requested from the cache at the same time. Due to this, I had to disable cache updates during runtime and only run the updates after the calculation was done. A possibility of enabling this in the future would include some sort of locking of cache during updates, but it would also slow it down considerably. The runtime cache remains in the code, but it is commented out.



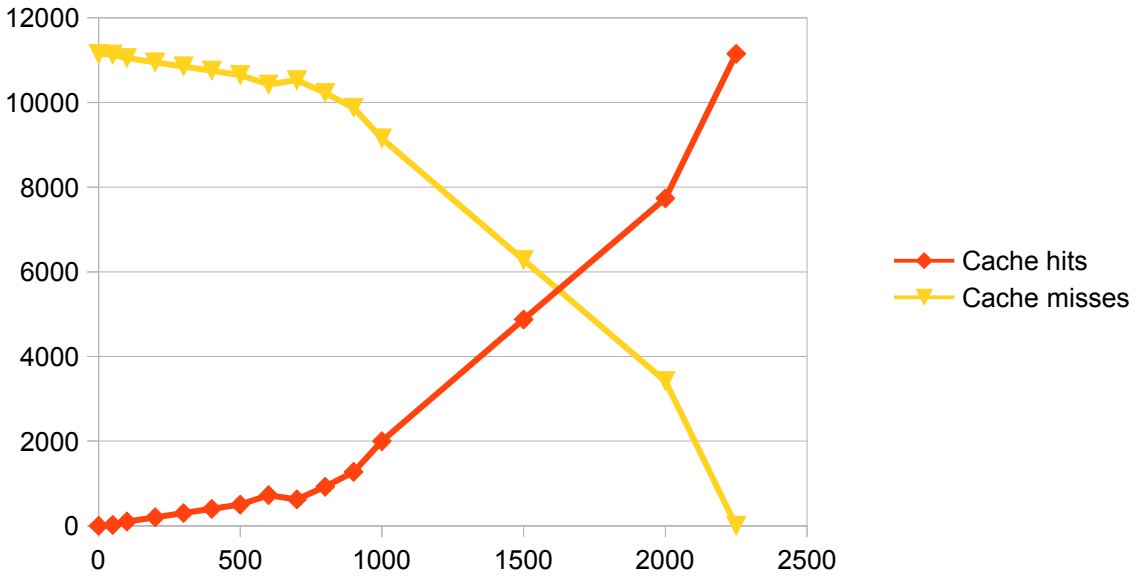
Picture 1, Plot problems caused by a misbehaving cache

Besides cache and its controls, I also implemented the Simplify button to the UI, that checked cache for calculations and replaced matching calculations one at the time. This worked rather well and works also with calculations like 3+2*3-12*sin(x).

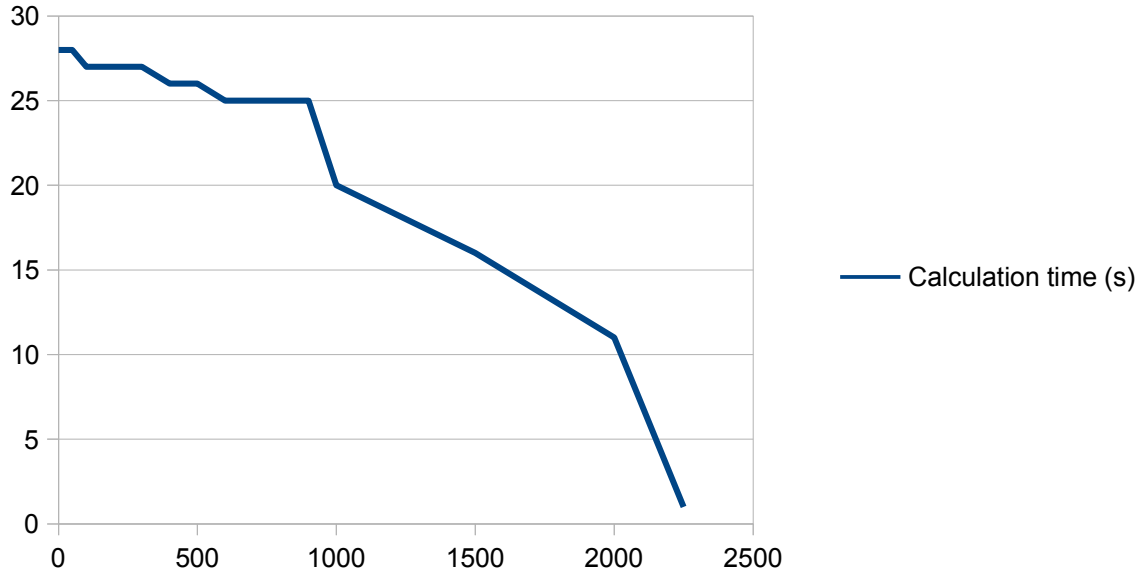
3. Plotting time of sin(x) vs. cache size

Tests were made with a laptop connected to wireless network (802.11n) with a average ICMP ping time of 3,733 ms and 12 hops between the client and server.

Tests were run for 15 different cache sizes *(0, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1500, 2000 & 2250)*, running the calculation once to have the results cached and then running it again. Each test was run 3 times to rule out extra network delays and median selected into data. Results are presented in pictures 2 and 3.



Picture 2, Cache hits & misses vs. cache size



Picture 3, Calculation time vs cache size

Data collected shows clear reduction in calculation time around cache size of 1000 calculations. With cache size of 2250, every item can be stored on a cache, reducing calculation time to a fraction of a second as no requests to the server need to be made.