

AutoPerf

Project by

- Aditya Kusupati
- Ritesh Theranikal

Guide

- Prof. Varsha Apte

Introduction

AutoPerf: An Automated Load Generator and Performance Measurement Tool for Multi-tier Software Systems

AutoPerf requires minimal input and configuration from the user, and produces a comprehensive capacity analysis as well as server-side resource usage profile of a Web-based distributed system, in an automated fashion.



Timeline of work

- **Literature**
- **Stability and Scalability:**
 - Capacity Analysis mode
 - Multi node mode
- **Profiler: (Major work)**
 - Server side Profiler
 - Client side (AutoPerf) output
- **More Experiments:**
 - Discovery of problems



Stability & Scalability


Making autoperf stable and scalable

Initial Problems

- AutoPerf used an algorithm to predict the next load level. That depends on :
 - Average response time of previous load test
 - Think time
 - Bottleneck CPU service demand that was estimated in that load test
- Generally whenever $N_users > 10,000$, then the next predicted load level was shooting out to 1,00,000 users
- The clients which we use for testing had no capacity of handling 1,00,000 threads
- This lead to crash of autoperf while it tried to initialise the worker threads



Solution or Fix


- We have discovered that most of the systems had a maximum limit on number of processes running at a single point of time and that can be changed, that is **ulimit**
 - For the clients we used for experiment maximum ulimit was 60,000 or 30,000
 - **Assumption:**
 - Around 10% of ulimit processes are already running
 - This can be fixed by parsing data in /proc/
 - When we find some load level prediction greater than 90% of ulimit, we fix it at the max limit we assumed and start the binary search for obtaining data
 - In our experiments most of the times we had saturation Around 10k users so we couldn't check for further robustness
- 

Multi-node version

- Autoperf has a multi-node mode which helps to scale up the client side user emulation
- When the single node version of AutoPerf was made stable, it was translated to Multi-node version
- Now we can emulate great number of threads for Load generation

Note: Most of the coding here was done by the project staff even though the ideas were conceived by us and we assisted him

Code was humongous and it took us good number of days even to get a hold of code flow



Profiler

Making profiler robust and fix bugs.
This was the major part of the RnD

Profiler

- Profiler is the server side code which is executed and gives out the results for requests fired from AutoPerf on the client side
- Profiler mainly is a code which extracts various system data using different systems calls
- Initially AutoPerf was giving wrong output for various fields in the expected output of AutoPerf
- **Deliverable:** Flawless Profiler and major fix of Autoperf profile output



Key functions

- getPSSnapShot
 - getMemorySnapshot
 - getDiskSnapshot
 - getNetworkSnapshot
 - getCPUUtilization
 - getCPUFrequency
- savePSSnapShot
 - saveMemorySnapshot
 - saveDiskSnapshot
 - saveNetworkSnapshot
 - saveCPUUtilization
 - saveCPUFrequency

These were the functions which delivered the data required for AutoPerf to generate Report.



Standard Experiment

We have used a pretty standard experiment to prove or disprove correctness of various measure given in AutoPerf Output

We made some files of known content and of various sizes.

Scripted multiple downloads using **wget** requests from the server.

Used the file client(single and multiple threads) we made in OS course.

Cross checked the output values with that of the profiler functions.



Results

- Profiler is really robust is correct in all aspects of the major functions described earlier
- All the outputs given by the ReportGenerator of Autoperf were fine except for:
 - Network Packets written (ie., sent)
 - Disk Blocks Read
- Code was decently big and we had to run multiple experiments on multiple server client setups to be sure of the experimental results

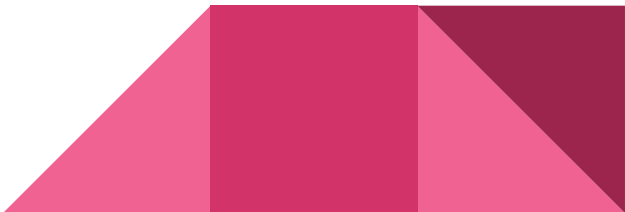


Problem in Network and Disk

- Network packets written is the actual sent packets on eth0 which has MTU of 1500 B
- So let N be the number of packets written, k be number of urls fired, M in Bytes be the size of file requested
- $N*1500 = k*M$, but we had $N*1500 < k*M$ and is $N*1500 \sim (k-a)*M$ where $a \geq 1$ $a < k$ and $a \propto n$ (number of users)
- We ran multiple experiments and were not able to figure out the problem
- Coming to Disk, we were said that it give 0-4 value most of the time, and rest of the time also we were not getting right answer



Solution and fix for Network

- We figured out from the previous explanation that few urls may be missed from the aggregate
 - We started off with checkpoints to figure out the error in the code flow
 - Initialisation was happening after thread creation has started this leads to missing of few urls at the starting
 - It is proportional to the number of users as initialisation is proportional to number of user threads
 - We made a code change to ensure initialisation happens before thread url firing
- 

Code Flow

Profile.java:

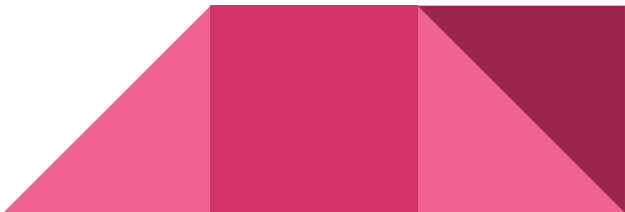
This program is responsible for creating all the worker threads.

Each worker thread independently starts firing url's to the server and downloads the associated file.

The duration of the experiment (during which the url's are fired) are specified in "mll.txt". For our experiment we've used a standard duration of 10 seconds.

The parent thread then continues to execute and creates the "MonitorRun" thread.

This thread's main function is to initialise and start the profiler and assimilate its results.



Code Flow

MonitorRun.java:

This code is responsible for starting the profiler and accepting the results from the server side.

MasterNetworkInterface.java:

Responsible for issuing various requests to the server to get information about the usage of various resources like disk, network bandwidth, memory, etc. using the functions listed earlier.

It saves the snapshots of the various resources, fires the url's and gets the snapshots again. The resource usage is inferred from the change in the values in the snapshot.

This is the main problem with the code flow. The concurrency in execution between the worker threads and the parent thread results in having some url's be fired before the profiler is started and the snapshot is acquired.

Hence, the results of the first few url's aren't incorporated into the results.

Disk

- Once we corrected the problem of the serialisation of the functions, we analysed the output of the disk.
- We cleared the cache on the server side to remove the file contents from the disk buffer cache.
- The same experiment was run on various file sizes, and the number of bytes read from the disk was observed in each run.
- We found that the output by the profiler for the disk usage was off from the file size by a constant factor in each instance, about a 1000.
- The factor is probably due to the granularity of the measurement, perhaps the block size.
- With cached data we get almost zero disk blocks as output

More Experiments

Discovery of problems

Problems

- AutoPerf is not saturating a definitely saturable server with single node on 2 experiment servers we used and is doing the saturation for the other 2
- A multi-node simulation of autoperf for the first case leads to saturation
- This lead to an unexplained problem, which we expect it to be a system related problem on the server side
- The problem only persists if there is a Database involved in the server side application



Future Work

Direction for future

Future Work

- Multi node scaling is a manually configured mode, this can be automated using a good heuristic obtained from theory and experimentation
- Need to solve the problem of not saturating servers: find a plausible explanation for the phenomenon
- Might consider using light weight threads instead of conventional threads
 - Quasar library



Rough Timeline of work

January

- Literature
- Getting started on code
- Minor bug fixes
- AutoPerf Stability using a manual limit based on system info

Feb - Mar

- Multinode
- Experiments to check profiler correctness
- Getting back onto track after going through a wrong explanation of facts
- Discovery of Profiler Problem of Network and Disk
- Fixing of the problem

April

- Robust testing after bug fix
- Explanation of Disk outputs
- Confirming Profiler is perfect
- Discovery of server dependent problem in scaling on single node
- More experiments

Contributions from other people

- Ravi Ranjan : Project Staff
 - Was the person who coded and resolved the scalability and stability issue for AutoPerf in both modes
- Akhilesh : RA
 - Helped us in performing experiments and collecting data and rechecking the work done through the experiments
- Devidas : Project Staff
 - Helped us in knowing the server dependent problem



Thank you

Varsha ma'am has been supportive all throughout the sem and understood our academic commitments. She guided us through the project with enthusiasm which encouraged us to work.