CS423 Spring 2016 MP4
Dynamic Load Balancer
Due April 25th at 11:59 pm 2016

1. Goals and Overview

1. In this MP you will design a Dynamic Load Balancer architecture for a Distributed System
2. You will learn and implement system monitors, adaptation and load balancing schemes to improve the system utilization and performance of a distributed application under limited bandwidth, memory and transient CPU load constraints.

2. Development Setup

For this MP you will develop a Dynamic Load Balancer for Linux. You will work entirely in the Linux user-space using Virtual Machines. The development setup will simulate a distributed infrastructure in which a local small factor device (e.g. cell phone or tablet) delegates some of its processing load to a remote higher-end computer.

For this MP you can choose between **Java, Python and C/C++.** You will use the provided VM as the remote higher-end computer (i.e. Remote Node), and also you are provided with an additional Virtual Machine as the local small factor device (i.e. Local Node). Your group's local node is named as **sp16-cs423-s-gxx.cs.illinois.edu**, where xx is your group number. Say you are group 1, your local node will be **sp16-cs423-s-g01.cs.illinois.edu**. You can turn on/off both VMs using the vSphere web client.

Finally, you are encouraged to discuss design ideas and bugs in Piazza. Piazza is a great tool for collective learning. **However, please refrain from posting large amounts of code.** Two or three lines of code is fine. High-level pseudo-code is also fine.

3. Introduction

The availability of low-cost microprocessor and the increased bandwidth of network communications have favored the development of distributed systems. However, one common problem of these systems is that in many cases some nodes remain underutilized while other nodes remain highly utilized. Maintaining even utilization in these systems is useful to increase system efficiency, performance and reduce power consumption.

One common technique to achieve even utilization across all nodes is using dynamic load balancing techniques that actively transfer the load from highly utilized nodes to lower utilization nodes.

This type of algorithm usually has 4 main components:

1. **Transfer policy**: Determines when a node should initiate a load transfer
2. **Selection policy**: Determines which jobs will be transferred.
3. **Location policy**: Determines which node is a suitable partner to transfer the load to or from. Please note that for our MP this policy is trivial, because there are only two nodes in the system.
4. **State Information policy**: Determines how often the state information about each node should be shared with other nodes.

4. Problem Description

In this MP, you will develop an architecture that implements a **dynamic load balance algorithm**. You will decide a suitable transfer, selection, location and information policies, evaluate and justify your decisions.

You will be required to implement a minimal set of components and functionality and you will have the opportunity to further augment and improve your design. You are encouraged to evaluate the performance impact of your decisions and to present your results as part of your demonstration.

The system architecture is composed of 2 nodes: a limited CPU Mobile node and a high-end Server node. These two nodes work together to compute a vector addition task. **You need to allocate a double vector with 1024 * 1024 * 4 elements named A.** For vector A, initialize all elements with 1.111111. For vector A, your task is to do the following computation (**DO NOT optimize the following computation**):

```
for(int i=0; i < 1024*1024*4; i++){
        for( int j=0; j < 6000; j++){
                A[i] += 1.111111;
        }
}
```

If you are using Python, perform an equivalent computation for vector A as follows:

```
for(int i=0; i < 1024*1024*4; i++){
        for( int j=0; j < 200; j++){
                A[i] += 1.111111;
        }
}
```

Note that the workload here is fully parallelizable. Your system should divide the workload into smaller chunks, called jobs. Each job is independent of each other, and each job should take roughly the similar time to finish. It's up to you to decide the size of the jobs during the implementation. However, initially there should be **at least 512 jobs**. During the transfer of the jobs, you MUST transfer the real data within vector A. Say one wants to compute vector addition from index 1000 to 1999. You must transfer the data within A[1000-1999] to the other node to perform the computation. That is, you must transfer 1000 elements, which is 8000 bytes (assume double occupies 8 bytes in the system), to the other node.

Our system will have three phases:

1) **Bootstrap Phase**: Originally, the Local Node will contain all the workload and it must transfer half of the workload to the Remote Node. This is called the bootstrap process. You **must** print out enough information (say, the job ID) on screen so that we can tell you are indeed transferring the jobs.

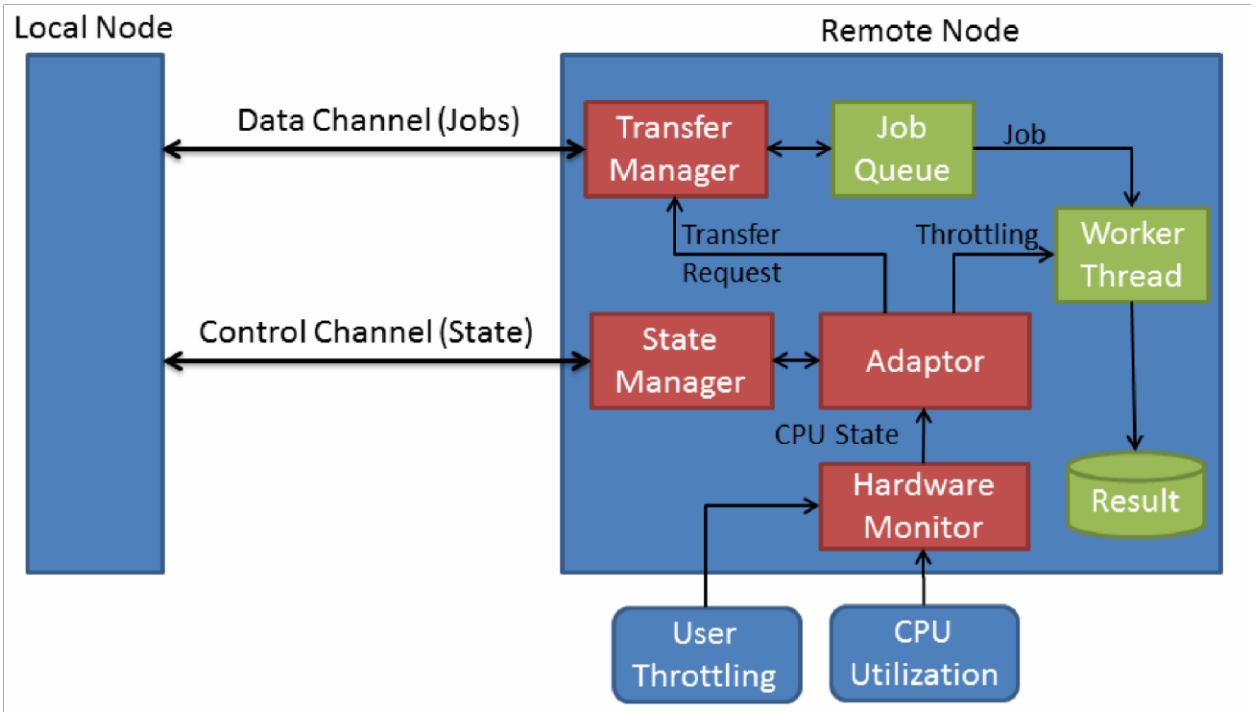
2) **Processing Phase**: After the bootstrap phase, each node divides the half workload into chunks

of data called jobs; each node will have a queue with roughly the same number of jobs. At this point, each node will start processing each of these jobs one by one and storing the result locally. Our dynamic load balancer will only work during this processing phase (i.e. by applying the transfer, selection, location and state information policies).

3) **Aggregation Phase**: After all the jobs are successfully processed, the system must aggregate the result into a single node (either local or remote) and display the result (prints on screen or writes to a file).

5. Implementation Overview

In this section, we present the minimal architecture that you must implement. Below you will find a figure showing the components of this architecture:



- **Job Queue**: The Job Queue is responsible of storing the pending jobs and dispatching them to the work thread.
- **Worker Thread**: The Worker Thread is responsible of processing each job and locally storing the result. The worker thread must react to throttling. This means, the worker thread must limit its utilization to a value provided by the Adaptor/User. For example, a throttling value of 70% means that during each 100 milliseconds, the worker thread must be sleeping for 30 milliseconds and processing the job for 70 milliseconds. During the sleeping period, the worker thread must not spin on a variable. You must use adequate system mechanisms to put the thread to sleep.
- **Hardware Monitor**: The Hardware Monitor is responsible for collecting information about the hardware. You are required to collect information about the CPU utilization of the system.

  However, as noted in the next Section you can also collect network bandwidth and other system information used by any of the policies of the Load Balancer (Transfer Policy,

Information policy and Selection policy). As part of the hardware monitor, you **must** implement an interface that allows the user to dynamically specify a throttling value to use to limit the worker thread during the execution. During the demo, we'll try different throttling values (like 0.1, 0.25, 0.5, 0.75) to limit the usage of either the local node or the remote node (or both).

- **Transfer Manager**: The Transfer Manager is responsible of performing a load transfer upon the request of the adaptor. It must move jobs from the Job Queue and send them to remote node. It must also receive any jobs sent by the remote node and place them in the local Job Queue. You can use any protocol that you choose (e.g. TCP, UDP, HTTP, etc.). You **MUST** print out enough info on screen (like job ID) when transferring and receiving jobs, so that during the demo we can tell that the jobs are indeed being transferred between the two nodes.
- **State Manager**: The State Manager is responsible of transferring system state to the remote node, including but not limited to, the number of jobs pending in the queue, current local throttling values, and all the information collected by the Hardware Monitor. For this component, you will need to choose an Information Policy. This policy will dictate how often the collected information is exchanged including time intervals or events. Careful design of this policy is important for system performance, stability and overhead tradeoffs.
- **Adaptor**: The Adaptor is responsible for applying the Transfer and Selection policies. It uses information provided by the State Manager and the Hardware Monitor, and decides whether a load balance transfer should occur. Please note that the most basic policy one can implement is to consider the Queue Length or the Estimated Completion Time (based on the throttling value and the queue length). A more sophisticated policy should use the Queue Length or the Estimated Completion Time and possibly other factors. Implementing a sophisticated policy is not required in the MP.

  As part of the implementation of your transfer policy, you must implement policies including sender-initiated transfers, receiver-initiated transfers and symmetric initiated transfers. Each of these has different tradeoffs in terms of performance, overhead and stability. **Analyze** what you have observed (job finish time, total number of jobs being transferred, total amount of data being transferred, etc) of choosing between sender-initiated transfers, receiver-initiated transfers or symmetric initiated transfers in the final submitted doc.

6. Further Improvements

Interested students are welcome to improve your MP in the following aspects. Note that all materials in this section are **not required** for this MP, and you will **not get extra points** for implementing the following improvements. Some of the improvements are as follows:

- You can extend the Hardware Monitor to consider Bandwidth and Delay. Assume the communication between the Local Node and the Remote Node is limited to 10 Mbit/s with average delay of 54ms (Std. Dev. 32ms) and average packet loss of 0.2%. Use this information as part of your Transfer Policy. Limited bandwidth can make load balance a very costly operation and further degrade the performance.

- You can compress the data when sending it over the internet to save time and bandwidth.
- You can implement a Graphical User Interface that shows the progress of your job and also clearly shows when your system initiates a transfer and changes a throttling value.
- You can increase the number of Worker Threads to exploit the concurrency of the system

7. Software Engineering

As in previous MPs, your code should include comments where appropriate. It is not a good idea to repeat what the function does using pseudo-code, but instead, provide a high-level overview of the function including any preconditions and post-conditions of the algorithm. Some functions might have as few as one line comments, while some others might have a longer paragraph. Also, your code must follow good programming practices, including small functions and good naming conventions.

For this MP you should choose adequate data structures to use. C++ and Java provide a large number of data structures with various performance characteristics. During the interview, you should justify your data structure design decisions.

8. Hand-In

All MPs will be collected using compass2g. You need to submit a single group_ID_MP4.zip (replace ID with your group number. If the filename is incorrect, -10 pts) file through compass2g. Only one of the group members need to do the submission. You need to provide all your source code (*.c and *.h) and a Makefile that compiles your code if applicable. Do not include binaries or other automatically generated files.

Finally, you must write a document briefly describing your implementation and design decisions. **You must include all the analysis that influenced your design decisions, including any experimental evaluation that you might have performed.**

In addition to the submission, you should be able to demo your source code and answer questions during an interview with the TAs. This interview will be a major part of the grading. Instructions for how to sign-up for the slot interviews will be posted at a later time on Piazza.

9. Grading Criteria

Criterion	Points
• **Bootstrap Phase**	**10**
• **Aggregation Phase**	**10**
• **Adaptor including Selection and Transfer Policy**	**15**
• **Transfer Manager**	**10**
• **State Manager**	**10**
• **Hardware Monitor**	**10**
• **Job Queue**	**10**
• **Efficient use of Locks Synchronization Primitives and Conditional Variables**	**5**
• **Code Compiles Successfully**	**5**

• **Code Follows Good Software Engineering Principles**	**5**
• **Documentation**	**10**
TOTAL	**100 / 100**

10. References

[1] N.G. Shivaratri, P. Krueger, and M. Singhal, "Load Distributing for Locally Distributed Systems", presented at IEEE Computer, 1992, pp.33-44.