Misco: A MapReduce Framework for Mobile Systems - Start of the Ambient Cloud?

Wednesday, August 18, 2010 at 7:03AM

Todd Hoff in Paper, ambient

Misco: A MapReduce Framework for Mobile Systems is a very exciting paper to me because it's really one of the first explorations of some of the ideas in Building Super Scalable Systems: Blade Runner Meets Autonomic Computing in the Ambient Cloud. What they are trying to do is efficiently distribute work across a set cellphones using a now familiar MapReduce interface. Usually we think of MapReduce as working across large data center hosted clusters. Here, the cluster nodes are cellphones not contained in any data center, but compute nodes potentially distributed everywhere.

I talked with Adam Dou, one of the paper's authors, and he said they don't see cellphone clusters replacing dedicated computer clusters, primarily because of the **power required** for both network communication and the map-reduce computations. Large multi-terabyte jobs aren't in the cards...yet. Adam estimates computationally that cellphones are performing similarly to desktops of ten years ago. Instead, they want to focus on the unique characteristics of the mobile devices--camera, microphone, GPS and other directly collectable data--so the data can be processed where collected.

MapReduce was selected as the programming interface because it is **familiar to programmers**, it transparently supports programming multiple devices, and can be implemented--especially using Python---in such a way that programmers are freed from all the underlying details like concurrency, data distribution, and code management. A very smart move.

It's interesting to contrast the economics of the ambient cloud to the economics of the data center cloud. The goal of a data center cloud is **100 percent utilization**. Use every possible CPU cycle or money is being wasted money on unused equipment. In an ambient cloud the idea is more parasitic, deploy to more resources yet leave the primary function of the device unaffected. It's a different perspective that may lead to different architectures. MapReduce scales linearly with the number devices. Since there are more phones than computers, using more of less capable devices will increase overall performance.

Misco first began as an intern project in the summer of 2008 at Nokia Research Center in Palo Alto. A quick introduction to Misco from the abstract:

> *The proliferation of increasingly powerful, ubiquitous mobile devices has created a new and powerful sensing and computational environment. Software development and application deployment in such distributed mobile settings is especially challenging due to issues of failures, concurrency, and lack of easy programming models. We present a framework which provides a powerful software abstraction that hides many of such complexities from the application developer. We design and implement a mobile MapReduce framework targeted at any device which supports Python and*

network connectivity. We have implemented our system on a testbed of Nokia N95 8GB smartphones and demonstrated the feasibility and performance of our approach.

An overview of the architecture is depicted by this excellent diagram:

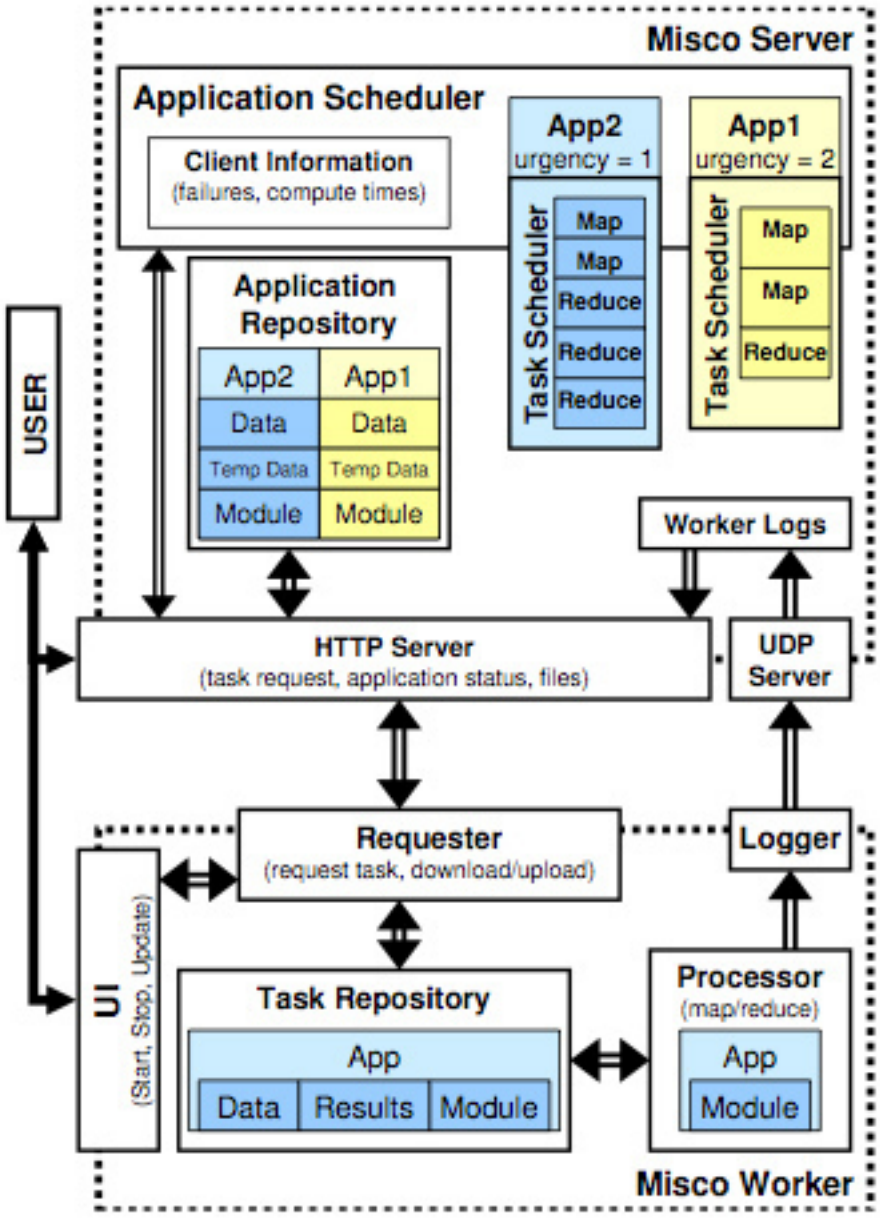

Figure 2: Architecture of the Misco System.

1. **MasterServer** - keeps track of user applications; maintains input,

intermediary and results data associated with applications; tracks worker progress; assigns tasks to workers. Communication between the server and worker is via HTTP. A server consists of:

1. **Application Repository Scheduler component** - keeps track of application input and output data.
2. **HTTP Server Scheduler component** - acts as the main communication channel between the server and workers. It also displays application status via a UI.
3. **UDP Server** - listens for incoming worker logs which are stored in Worker Logs.

2. **WorkerNode** - performs map and reduce operations and returns the results to the server. Can be run on any computing device that supports Python. A worker consists of:
    1. **Requester component** - interacts with the server to requests tasks; uploads and downloads data; triggers local task execution; performs upgrades.
    2. **Repository component** - stores the input data, modules, and results for each task.
    3. **Logger component** - maintains local process times and progress and uploads them to the server along with the results when a task completes.
3. **Polling** - workers poll servers for work. Push notification is not generally supported so polling is more reliable, though it can be difficult to tune the polling interview correctly.
4. **General Process**:
    1. A job is created with the server and the server schedules the job. A job consists of the deadline, input data, module, size of the map input pieces, and number of reduce partitions.
    2. The server splits the input data into M inputs files.
    3. Workers make work requests to the server.
    4. Workers submit the results to the server.
    5. Once all the map results have been received and similar

process occurs for the reduce phase.

Seems like a very clean and logical architecture.

Experiment Setup

The test is run on 30 Nokia N95 8GB smart-phones, which have an ARM 11 dual CPUs at 332 Mhz, 90 MB of main memory, 8GB local storage. The server was a Pentium 4 2Ghz CPU with 640 MB of memory. Phones were connected to a router using 802.11g and the router connected to the server using a 100 MBit connection.

They implemented a WordSearch application to search for keywords in a large text file. Your common MapReduce example problem. The goal is to interactively search web logs to find places of interest within a city. Input files ranged from 10KB to 1MB.

Results

- The Misco library took 800KB of memory, or 1% of the available 90MB.
- A redundancy scheme was used to handle failures, as failures increased the end-to-end times increased exponentially.
- Memory usage varies by the type of application. If the application uses 15MB of data then the total used will be 16MB.
- Processing tasks used .7 watts.
- Network access used 1.6 watts. Cell networks use more energy than WiFi.
- Majority of processing time is spent performing computations, only a small fractions is spent on overhead.

The dominance of power usage of network transfers over local computation may dictate where optimization efforts go in the future. As

an example of this dynamic Adam points to the runtimes for the Kindle with and without network connection and with or without 3G.

Adam suggests a lot of energy may be saved by compressing the data so that there is much less data to send. Another approach is to pre-process the data on the phone. Say you are are performing and image recognition task on pictures that are already on the phone. It's possible to extract vector features from the pictures, which is a much more compact representation, with the end result being far less data transferred.

Batching data would use less network resources, but that must be balanced against responsiveness. How would using the cell network versus WiFi impact performance or power usage? For devices that support push more efficiently, would power usage go down significantly? Python is a very practical deployment language, but is the typical 20% overhead for a dynamic language the best approach for a cellphone? Lot's of things to think about.

Obviously still early days, but it's exciting to see the beginning of this kind of research. Adam promises a lot more in the future.

Related Articles

- Exploring How to Build a Cloud With Smartphones by Alex Williams of ReadWriteWeb.
- Misco Home Page
- Disco - a distributed computing framework based on the MapReduce.
