Lecture 12: Cloud Computing Fundamentals (10/28 + 10/30)

You can use this AWS tool to get an estimate about the potential cost of your cloud resources: https://calculator.aws/#/

We did a back-of-the-envelop calculation to estimate how much Dropbox needs to pay to move 500PB of data out of AWS S3: It turns out to be approx. 25 Million dollars!

Breaking cloud lock-in barrier:

Researchers and practitioners are constantly looking for solutions to overcome cloud lock-in barrier. One current effort is a new computing paradigm called **Sky Computing**. Sky Computing proposes mechanisms to protect users from major cloud lock-in or major cloud outages, by imposing uniform and comprehensive standards, but by creating a fine-grained two-sided market via an intercloud broker.

Take a look at the SkyPilot paper:
https://www.usenix.org/conference/nsdi23/presentation/yang-zongheng

Follow up work of SkyPlane, which is an efficient intercloud data transfer system:
https://www.usenix.org/conference/nsdi23/presentation/jain

Lecture 9: Consistency and Linearizability (09/30 + 10/02)

Linearizability is the strongest form of consistency that one (distributed) system could achieve. As discussed in lecture, linearizability requires real-time ordering of requests. One may ask, how one different computers perfectly synchronize their local clocks. If you recall, as we discussed in “Logical Clock”, synchronizing clocks using naive approaches (e.g., exchanging timestamps) is prone to significant accuracy loss.

One way to achieve real-time ordering is by processing requests on a single computer, e.g., a leader. The leader can use its local clock to real-time order requests that are being processed concurrently.

A more advanced technique of achieving (*approximate*) real-time-ordering is by using an almost-perfectly sync'ed global clock, called TrueTime (see Google Spanner paper and a discussion of Spanner's TrueTime). TrueTime is a global synchronized clock with bounded non-zero error: it returns a time interval that is guaranteed to contain the clock's actual time for some time during the call's execution. Thus, if two intervals do not overlap, then we know calls were definitely ordered in real time. If the intervals overlap, we do not know the actual order.

The underlying time references used by TrueTime are GPS and atomic clocks [Spanner paper]. Spanner is claimed to achieve external consistency (similar to linearizability) from TrueTime. Spanner's external consistency invariant is that for any two transactions, T1 and T2 (even if on opposite sides of the globe):

if T2 starts to commit after T1 finishes committing, then the timestamp for T2 is greater than the timestamp for T1.

Lecture 7: GFS (09/18/24 + 09/23/24)

To download the HDFS binary:
wget https://dlcdn.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz

Several HDFS commands that you need to get familiar with:

hdfs namenode -format # to format an HDFS namenode
start-dfs.sh # to launch an HDFS cluster (including both NameNode and DataNodes)
jps # to check the running Java processes (of an HDFS deployment)

hdfs dfs -help # to print the usage of the cmd
hdfs dfsadmin -report # to print the status of HDFS cluster
hdfs dfs -copyFromLocal # copy a local file to HDFS
hdfs dfs -ls # list a dest dir in HDFS
hdfs dfs -cp # copy an HDFS file to another HDFS file
hdfs dfs -mkdir # create a dir in HDFS
hdfs dfs -du # check the disk utilization of an HDFS file
hdfs fsck # check the health status of an HDFS file

While CLI is a great tool, it turns out you can also browse the current status of your HDFS cluster using your browser:
<public_IPv4_DNS_address_of_vm1>:9870/dfshealth.html

You can explicitly configure the replication factor and block size of an HDFS file (-D allows you to change HDFS' Java-program-internal configurations):

hdfs dfs -D dfs.replication=1 -cp <file1> <file2>
hdfs dfs -D dfs.block.size=256m -cp <file1> <file2>

This document contains most of the Java-specific HDFS configuration options:
https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml

Lecture 6: RPC in Go and Lab 1 tutorial (09/16/24)

The Go RPC demo code:

```
package main

import (
      "fmt"
      "strings"
      "os"
      "net"
      "net/rpc"
)

type WordCountServer struct {
       addr string
}

type WordCountRequest struct {
       Input string
}

type WordCountReply struct {
       Counts map[string]int
}

func (*WordCountServer) Compute(
             request WordCountRequest,
             reply *WordCountReply) error {
       counts := make(map[string]int)
       input := request.Input
       tokens := strings.Fields(input)
       for _, t := range tokens {
              counts[t] += 1
       }
```

```
	reply.Counts = counts
	return nil
}

func checkError(err error) {
	if err != nil {
		fmt.Println("Error:", err)
		os.Exit(1)
	}
}

func (server *WordCountServer) Listen() {
	rpc.Register(server)
	listener, err := net.Listen("tcp", server.addr)
	checkError(err)
	go func() {
		rpc.Accept(listener)
	}()
}

func makeRequest(input string, serverAddr string) (map[string]int,
error) {
	client, err := rpc.Dial("tcp", serverAddr)
	checkError(err)
	args := WordCountRequest{input}
	reply := WordCountReply{make(map[string]int)}
	err = client.Call("WordCountServer.Compute", args, &reply)
	if err != nil {
		return nil, err
	}
	return reply.Counts, nil
}

func main() {
	serverAddr := "localhost:8888"
	server := WordCountServer{serverAddr}
	server.Listen()
	input1 := "hello I am good hello bye bye bye bye good night
hello"
	wordcount, err := makeRequest(input1, serverAddr)
	checkError(err)
	fmt.Printf("Result: %v\n", wordcount)
}
```

Lecture 4: MapReduce (09/09/24)

Resources on shared memory:
IVY: https://systems.cs.columbia.edu/ds2-class/papers/li-ivy.pdf
This paper discusses the differences between shared memory and message passing.

Practical modern implementation of a distributed shared memory (DSM) system:
Ray white paper:
https://docs.google.com/document/d/1tBw9A4j62ruI5omIJbMxly-la5w4q_TjyJgJL_jN2fI/preview#heading=h.iyrm5j2gcdoq
Ray implements a distributed in-memory object store, resembling a DSM system.