CS 1380

Day 2 : Jan 26

Theophilus Benson
1. Motivation for D.S. concepts (why is D.S. hard)
2. Networking (How do this enable D.S.)
3. Failures (types, practicality, overheads)
4. “Fallacies of distributed computing”
MapReduce = large scale + easily distribute + massive workloads

1. pick servers for mapping
2. for each mapper inform it of storage to use
3. pick server(s) for reduce
4. inform mappers of reduce location

Requirements
(a) keep track of all servers
   (i) health
   (ii) progress
   (iii) data location
   (iv) load utilization

map

reduce

DONE!!!
Count (*) from all data

reduce

where name = 'theo'
\& age \geq 12

map

\geq 2%

Why is the master node a bad idea?

① single point of failure
② all load is one server

① cheap
② easy to write
③ easy to coordinate
④ how much up time is good enough?

Does simplicity still matter with formal verification?
failure / performance problems

1. failure types
   a. network
   b. power
   c. resource exhaustion (CPU/MEM)
   d. restart (state failure)
   e. hardware failure

2. easy to detect
   a. server / power
   b. some HW failures

failure detection = heartbeat messages

Questions about designing
1. frequency
2. performance impact
3. msg size
4. much data to detect failure
1) How long should you wait for a reply?

- Wait one nanosecond
- Wait forever

2) How often should you send heartbeat messages?

- Every second
- Every two years

(1) Depends on CPU (speed/generation)!!
(2) Criticality of functionality!

Backup → Google search
Latency\n\n\text{Heartbeat} = n/w + OS + \text{process} + n/w \n\n\text{Heartbeat time is too small} \rightarrow \text{mistakes (where you think the server is down but it's up)} \n\n\text{Theo. Com}\n\begin{enumerate}
\item 100k servers
\item Power outage takes out 30k
\item After outage all servers are online
\end{enumerate}