ANNOUNCEMENTS

• Assignment 1 out
• CloudLab notes on Piazza
• No teams yet?
BACKGROUND: PTHREADS

```c
#include <pthread.h>

void *myThreadFun(void *vargp)
{
    sleep(1);
    printf("Hello World\n");
    return NULL;
}

int main()
{
    pthread_t thread_id_1, thread_id_2;
    pthread_create(&thread_id_1, NULL, myThreadFun, NULL);
    pthread_create(&thread_id_2, NULL, myThreadFun, NULL);
    pthread_join(thread_id_1, NULL);
    pthread_join(thread_id_2, NULL);
    exit(0);
}
```

- Fine-grained synchronization
- Locks
- Shared memory
int main(int argc, char** argv) {
    MPI_Init(NULL, NULL);
    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    // Print off a hello world message
    printf("Hello world from rank %d out of %d processors\n", world_rank, world_size);
    // Finalize the MPI environment.
    MPI_Finalize();
}
MOTIVATION

Build Google Web Search

- Crawl documents, build inverted indexes etc.

Need for

- automatic parallelization
- network, disk optimization
- handling of machine failures
OUTLINE

- Programming Model
- Execution Overview
- Fault Tolerance
- Optimizations
Data type: Each record is \((key, value)\)

**Map function:**

\((K_{in}, V_{in}) \rightarrow \text{list}(K_{inter}, V_{inter})\)

**Reduce function:**

\((K_{inter}, \text{list}(V_{inter})) \rightarrow \text{list}(K_{out}, V_{out})\)
def mapper(line):
    for word in line.split():
        output(word, 1)

def reducer(key, values):
    output(key, sum(values))
WORD COUNT EXECUTION

Input
- the quick brown fox
- the fox ate the mouse
- how now brown cow

Map
- Map
- Map
- Map

Shuffle & Sort
- the, 1
- brown, 1
- fox, 1
- how, 1
- now, 1

Reduce
- brown, 2
- fox, 2
- how, 1
- now, 1
- the, 3

Output
- ate, 1
- cow, 1
- mouse, 1
- quick, 1
ASSUMPTIONS
ASSUMPTIONS

1. Commodity networking, less bisection bandwidth
2. Failures are common
3. Local storage is cheap
4. Replicated FS

*compared to 1 machine*
WORD COUNT EXECUTION

Submit a Job → JobTracker

Automatically split work

Map

File, offset, size

the quick brown fox

Reducer

Map

Location, size

the fox ate the mouse

Map

Map

Schedule tasks with locality

Resource manager

map-reduce master

how now brown cow

Not necessary

Hadoop

wait all mappers
Fault Recovery

If a task crashes:

- Retry on another node
- If the same task repeatedly fails, end the job
If a node crashes:
- Relaunch its current tasks on other nodes
What about task inputs? File system replication

Fault Recovery

If a node crashes:
- Relaunch its current tasks on other nodes
What about task inputs? File system replication

Fault Recovery

Map
quick fox

Map
the fox ate the mouse

Map
the quick brown fox

Map
how now brown cow
If a task is going slowly (straggler):

- Launch second copy of task on another node
- Take the output of whichever finishes first
MORE DESIGN

Master failure $\equiv$ Do nothing!

Locality

Scheduling of Mappers/Reducers

Task Granularity
REFINEMENTS

- Combiner functions
- Counters
- Skipping bad records

FS itself have bad records

Don't want to fail job if 1 record out of $10^9$

Corner case

Answer is okay?

Stragglers?
# MapReduce Usage Statistics Over Time

<table>
<thead>
<tr>
<th></th>
<th>Aug, ‘04</th>
<th>Mar, ‘06</th>
<th>Sep, ’07</th>
<th>Sep, ’09</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of jobs</td>
<td>29K</td>
<td>171K</td>
<td>2,217K</td>
<td>3,467K</td>
</tr>
<tr>
<td>Average completion time (secs)</td>
<td>634</td>
<td>874</td>
<td>395</td>
<td>475</td>
</tr>
<tr>
<td>Machine years used</td>
<td>217</td>
<td>2,002</td>
<td>11,081</td>
<td>25,562</td>
</tr>
<tr>
<td>Input data read (TB)</td>
<td>3,288</td>
<td>52,254</td>
<td>403,152</td>
<td>544,130</td>
</tr>
<tr>
<td>Intermediate data (TB)</td>
<td>758</td>
<td>6,743</td>
<td>34,774</td>
<td>90,120</td>
</tr>
<tr>
<td>Output data written (TB)</td>
<td>193</td>
<td>2,970</td>
<td>14,018</td>
<td>57,520</td>
</tr>
<tr>
<td>Average worker machines</td>
<td>157</td>
<td>268</td>
<td>394</td>
<td>488</td>
</tr>
</tbody>
</table>

Jeff Dean, LADIS 2009
DISCUSSION

https://forms.gle/hK8wFDxBDfS6chD28
DISCUSSION

Indexing pipeline where you start with HTML documents. You want to index the documents after removing the most commonly occurring words.

1. Compute most common words.
2. Remove them and build the index.

What are the main shortcomings of using MapReduce?
DISCUSSION

(a) Normal execution  
(b) No backup tasks
NEXT STEPS

• Next lecture: Spark
• Assignment 1: Use Piazza!
• Project topics: End of this week