YAPP, A Data Processing Framework

Xilong Fan
xfan@spokeo.com

Challenges

Ever Growing Data Sheet {
- Data At the Size of Hundreds of Million Becomes Common, Billion for DP
- Raw File for Processing and Loading May Runs At Hundreds of GB Level
- Data Continued Being Pulled From Multiple Sources on Timely Basis.
}

Jobs Becomes More And More Time Sensitive {
- Data Is **LOST** if it Became **Dated** Already or **Cannot Be Retrieved**.
- Some Data Loading, Normalization Job is Monthly Based(Crime Data)
- Answer Ad-Hoc Queries from Marketing Team is Even More Challenging.
}

Manually Adjusting Processes IS Always Needed {
- Tens of Processes for Small Table Refresh is ok, **Ugly** When Num. > 128
- Always Need to Restart Proc Since There is **NO** Perfectly Distributed Load
- Manually Keeping Proc. Info is Guaranteed **Error Prone**, Always.
}

Possible Solutions for Parallel Processing

MapReduce Paradigm(Hadoop on HDFS) {
- The Most Popular De Facto Standard for Massive Parallel Crunching.
- Pros:
  - Enjoys the Benefit of Community With Top Developers.
  - Primarily Designed for Fault-Tolerance Based on Commodity Hardware
- Cons:
  - Specifically Optimized for Massive File IO Use Large Block Size, Not DB.
  - Needs HDFS, Also Require A Complete Codebase Rewrite.
  - Redundant Job Scheduling to Speedup Lagging tasks is not desirable.
  - Most I/O are MySQL related, causing the biggest resource limitation.
  - Not All Problems Could Fit into the MapReduce Paradigm(Graph…)

}

MPI on OpenPBS/Torque {
- The Most Widely Accepted Batching System for Research Institutions.
- Pros: FAST, Handy Proc. Coordination, Favored by HPC Community.
- Cons: Solely Designed for Clusters/Super Computers, No Fault-Tolerant.

}

Flexible, Fault-tolerant Yet Still Light Weighted Framework

YAPP(Yet Another Parallel Processing)

Overview {
- A Push Based(Master & Worker), Durable Job Scheduling System.
- Master Accepts the Job Submission, do the Splitting and Push to Worker
- Worker Initiates the Processes and Monitoring Their Running.
- Key Components for Yapp Service:
  - yapp: user use it to submit jobs to run in parallel
  - yappd: background instances actually do the job and process requests.
  - ypadmin: providing common job manage interface for user & admin.
- Implemented by C++, Using Thrift for RPC, Zookeeper for Coordination.
}

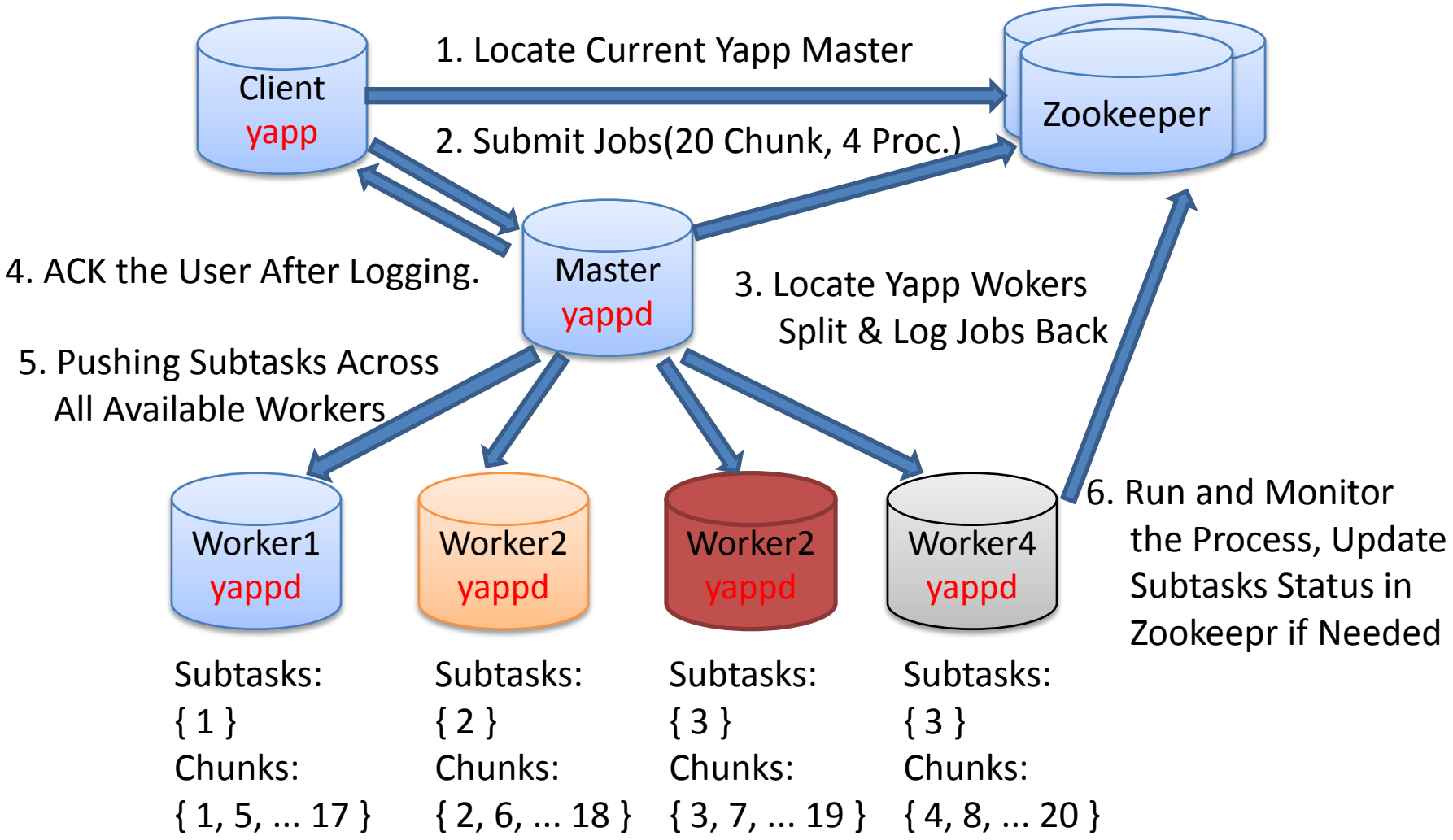
Features {
- User Decides How to Split the Tasks, How Many Proc. In cfg file to Run.
- User Could Log Current Anchor Pos. in Log File and Use it to Do Skipping.
- Yapp Service Will Guarantee ‘At Least Once’ Semantic for Each Subtasks.
- Once Submitted, Jobs Never Get Lost
}

YAPP(Yet Another Parallel Processing)

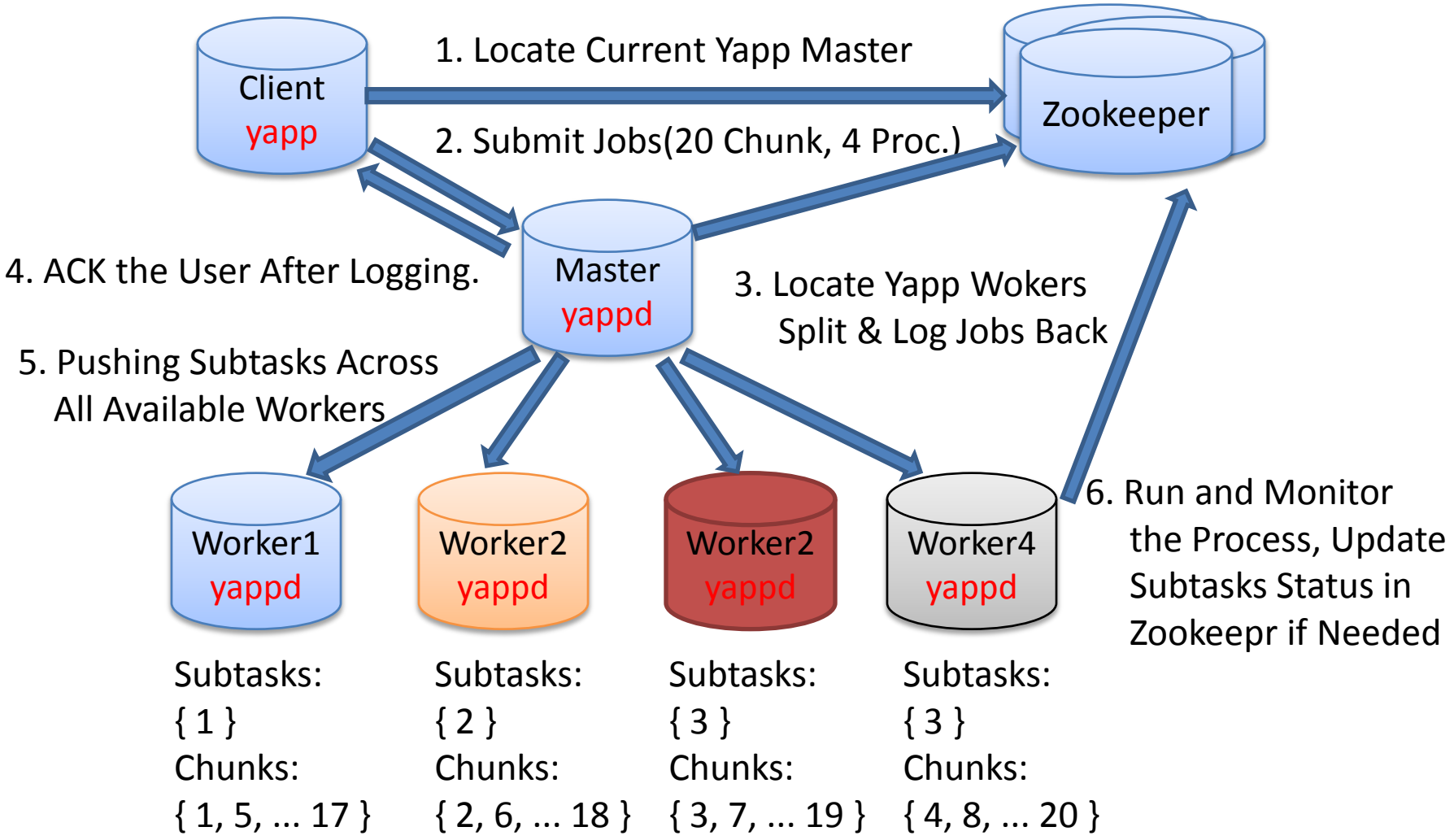
Why Thrift? {
- A Safe and Robust Solution Comparing to Raw Socket Programming
- Top Projects Under Apache, Actively Maintained and Widely Used.
- Come With Out of Box Thread Pooling Implementation.
}

Why Zookeeper? {
- A Replicated, Fault-Tolerant Key-Value DB With Simple File System API
- Use Quorum Based Protocol, Stands Up to F Failures for 2F + 1 Nodes
- Nice Feature of “Single System Image” per Client for Data Consistency.
- Supports Atomic Semantic for Operations Across Multiple Records
- Free Heart-Beat Implementation, Perfectly for Nodes Management.
- Supports EMPHEMERAL Record, Ideal for Distributed Locking.
- Meta Data for Each Subtasks Won’t be Huge.
- Widely Used By Hadoop/HDFS Projects Family.
}

A General Design and Work Flow



A General Design and Work Flow



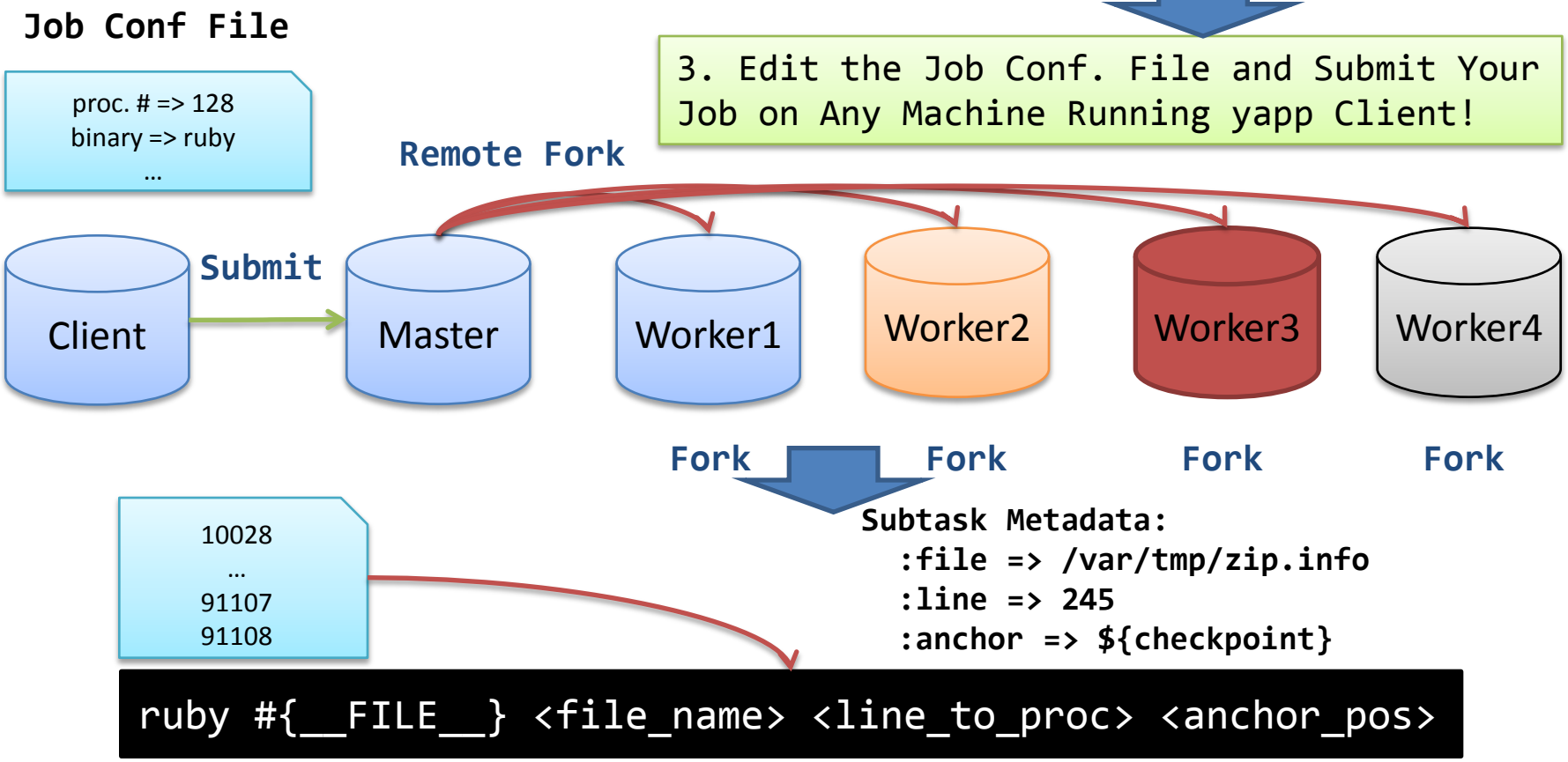
ALL yappd instances are created equal, any worker node can gain mastership if needed!

How My Jobs Got Farmed Out?

1. Generate A Job Metadata File Where Every Single Subtask is Specified in 1 line (Minimum Processing Unit, can be either a zip code or a path for a file chunk).

2. Your program to be farmed out should have at least three parameters reserved on command line interface, which will be used by YAPP to pass some task related configuration.

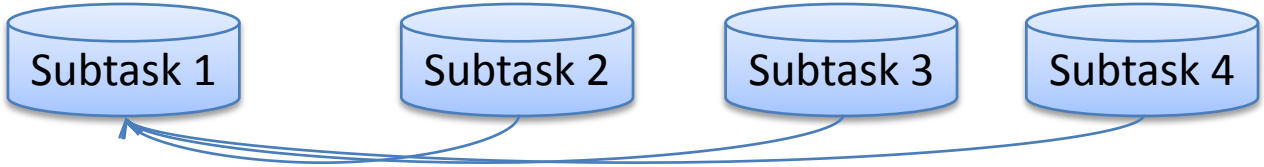
3. Edit the Job Conf. File and Submit Your Job on Any Machine Running yapp Client!



Handling for Un-Even Load(Most Likely)

What if 99% Subtasks Finished Their Own Processing Chunk in 1 Night?

Dynamic Load Rebalancing on Yapp



Processing 1 Chunks: { 1, 5, ... 17 }

Processing 5 Chunks Done { 2, 6, ... 18 }

Processing 9 Chunks Done: { 3, 7, ... 19 }

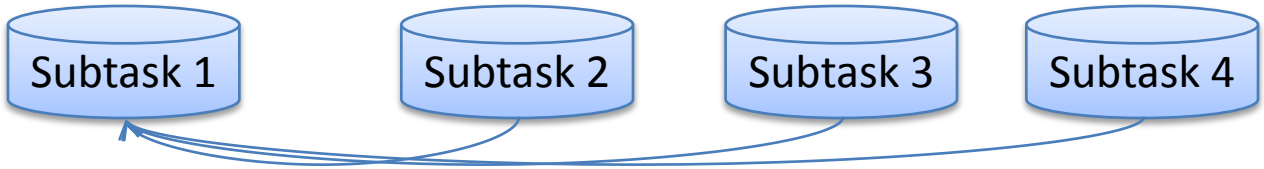
Processing 13 Chunks Done: { 4, 8, ... 20 }

Subtasks Will Not Idle and Sleep As Long As There Are Any Subtasks to Help

Handling for Un-Even Load(Most Likely)

What if 99% Subtasks Finished Their Own Processing Chunk in 1 Night?

Dynamic Load Rebalancing on Yapp

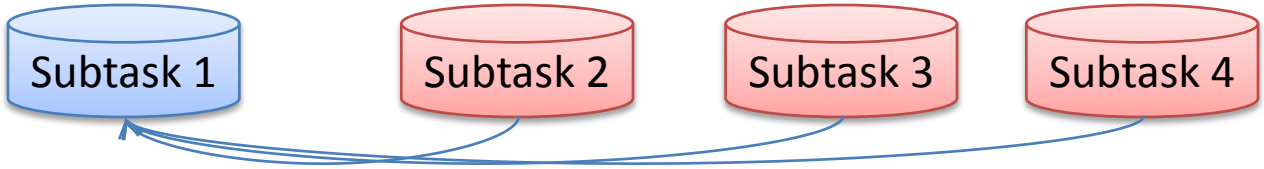


Processing 1 Chunks:	Processing 5 Chunks Done	Processing 9 Chunks Done:	Processing 13 Chunks Done:
{ 1, 5, ... 17 }	{ 2, 6, ... 18 }	{ 3, 7, ... 19 }	{ 4, 8, ... 20 }

Subtasks Will Not Idle and Sleep As Long As There Are Any Subtasks to Help

What if I Need to Restart Only One Failed Subtask While All Others Done?

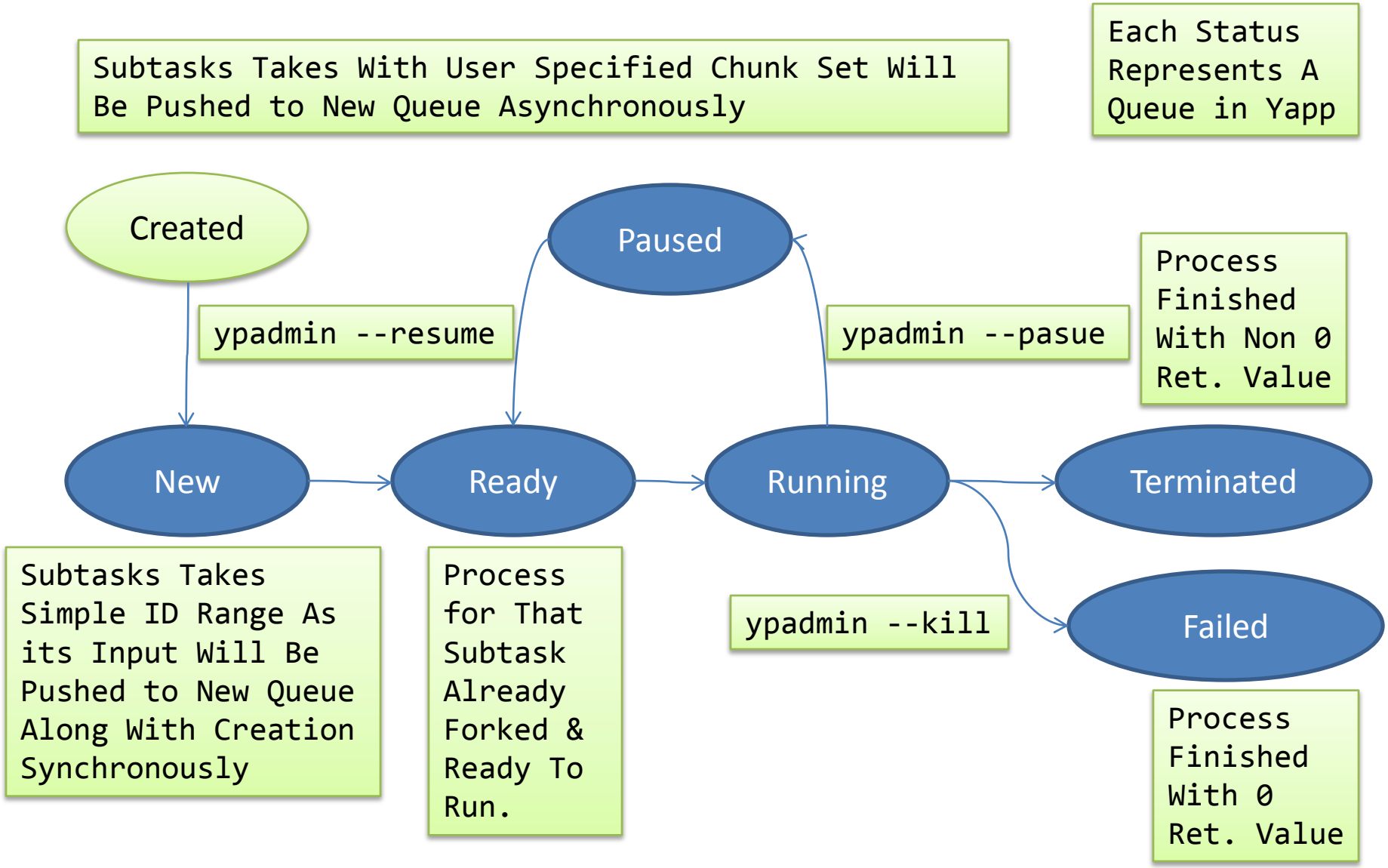
Subtasks Will Be Invoked (Requeued) Even if They Already Terminated



Processing 1 Chunks:	Processing 5 Chunks Done	Processing 9 Chunks Done:	Processing 13 Chunks Done:
{ 1, 5, ... 17 }	{ 2, 6, ... 18 }	{ 3, 7, ... 19 }	{ 4, 8, ... 20 }

Subtasks Will be Requeued As Long As The Thread on Master Feel A Need to Do This

SubTask Status Transition Among Queues



Sample Configuration for yappd

```
zkserver.1=192.168.1.109:2181
zkserver.2=192.168.1.111:2181
zkserver.3=192.168.1.117:2181

port=9527
thrd_pool_size=16
max_zkc_timeout=40000
max_queued_task=2048
mode=master
master_check_polling_rate_sec=30
pid_file="/var/run/yappd/yappd.pid“

# Specifies the time interval(sec.) for master to schedule the subtasks tasking
# range file as its input(push them to the NEW queue).
range_file_task_scheule_polling_rate_sec=5

# Specifies the time interval(sec.) for master to schedule the subtasks logged
# in the NEW queue.
subtask_scheule_polling_rate_sec=5

# Specifies the time interval(sec.) for the master to check if any subtask left
# as ZOMBIE(Master lost the Heart Beat to It's Running Node).
zombie_check_polling_rate_sec=60

# Specifies the time interval(sec.) for master to check if any subtask tasking
# range file as its input needs more proc.s to run.
rftask_autosplit_polling_rate_sec=10
```

Sample Conf. File for Job Submission Using yapp

```
--proc_num=120
--app_env=RAILS_ENV=production
--app_bin=/usr/bin/ruby
--working_dir=/home/src
--app_src=./foo.rb
--arg_str=
--range_file=./input/file-list.conf
--stdout=/var/tmp/foo.stdout
--stderr=/var/tmp/foo.stderr
--anchor_prfx=UPDATE_RA_CUR_ID_POS=
```

Utilities Supported By ypadmin

```
ypadmin --qstat | --print  (${job_hndl}|${task_hndl}|${proc_hndl})*
        --pause | --resume (${job_hndl}|${task_hndl}|${proc_hndl})*
        --restart-failed   (${job_hndl}|${task_hndl}|${proc_hndl})*
        --restart-full     (${job_hndl}|${task_hndl}|${proc_hndl})*
        --kill  | --purge  (${job_hndl})*
                      +--> PURGE ONLY WORKS WHEN ENTIRE JOB TERMINATED.
      { --init, --ls, --tree, --get, --rm, --rmtree, --create }
      ==>> Directly Accessing Zookeeper, Only Use It At YOUR OWN RISK
```

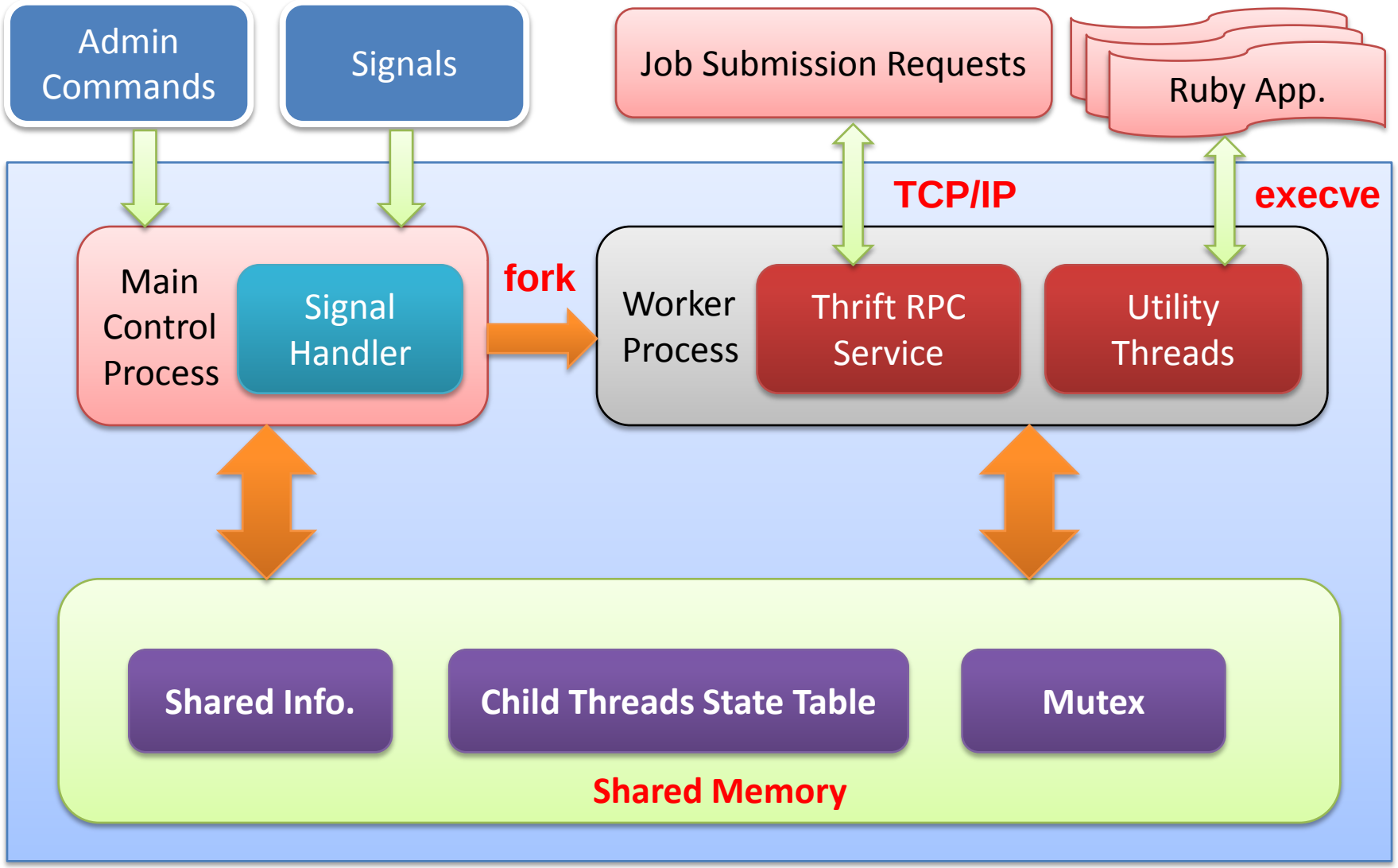
Yapp & MapReduce

The Same {
- All Use Push Mode, in which master farms out jobs to available workers
- All Use Zookeeper for Nodes Management and Metadata
- All Provide Durability for Job, Once Submitted, Never Lost
}

Differences {
- Yapp Only Needs to Do “Map”, Which Makes it Much More Simpler.
- Yapp Does Not Assume Single Master, Can Easily Switch to Multi-Master.
- Fully Functional Based On System Shell Instead of A Separate Library
- General Enough To Support Parallelization for Any Kinds of Program
- Solely designed for simplifying the human work of process management
- A Handy Tool for Easy Parallelization Based On Our Current Code Base
- The “All Created Equal” Architecture Makes it No Single Point of Failure
- Not designed as a library for massive parallel over millions of nodes
- Not Built On top of Any Distributed File System
}

Future Work

- Making Yapp More Clean & Reliable by Using Multi-Process Architecture



Thanks!