Distributed Processing in Clojure

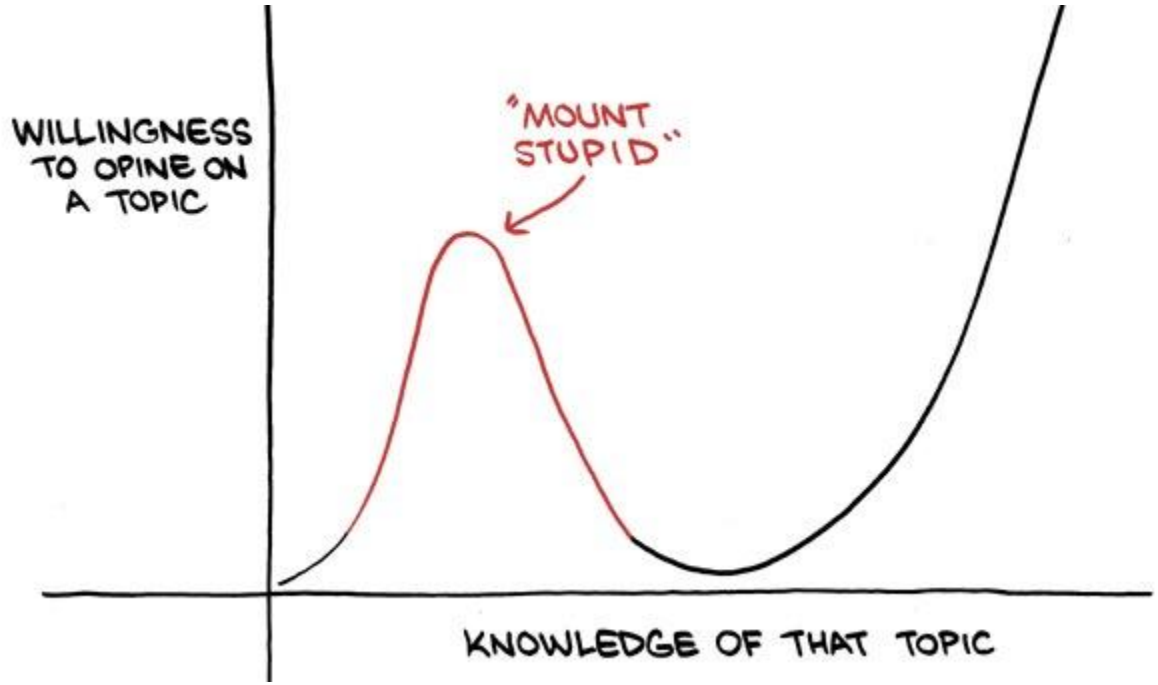
matt hoffman
@matt_thinks_so
www.matt-thinks-so.com

Who am I?

- Why am I talking about this?

- Why should you listen to me?

Who am I? cont.


WILLINGNESS TO OPINE ON A TOPIC
"MOUNT STUPID"
KNOWLEDGE OF THAT TOPIC

Who am I? cont.

- Fairly new to Clojure
  - ~1 year
- Not so new to distributed systems
  - ~10 years
    - DHS
    - Intelligence
    - Retail forecasting & analytics

Who are you?

Experience?

- How many have used Hadoop or Storm?
- Something else?

Questions to start out?

Things you're hoping to hear?

Why are we here?

- Recently evaluated distributed processing options
  - Requirements may or may not line up with yours...

- Seen a lot of questions like:

  “Hey I have this Clojure app and I need to distribute processing across a few machines, what do I use?"

some answers:

- "just use Cascalog"
- "just use Storm"
- "just use a queue"

Roadmap

Intro (we're in that)

3 Scenarios (data processing, event processing, everything else)

- What are some options?
  - what was it designed for?
  - What scenarios does it excel in?
  - What are the boundaries? Requirements?

Deeper dive into some of those (as needed?)

Intro to my project (if there's time)

"Distributed Processing"?

Not distributed data.

...necessarily

"...in Clojure?"

The JVM has a ton of options.

The talk's title is a lie.

Some Scenarios

- “I have a mountain of data, a room full of computers, and I'm ready for a good time”
  - *data processing*

- “I need to track the public's opinion of Congress in real time”
  - *event processing*

- “I have a web server, and need to push some async processing off my server”
- "I am a special snowflake, and my problems are unique"
  - *everything else*

- **1st Scenario: Process large amounts of data**

Scenario 1: Processing Data

- Hadoop and its kin own the space
  - there are contenders...
- Not "distributed processing" as much as "process large amounts of data".
- Don't use Hadoop directly. Use a layer on top:
  - Cascalog
  - Cascading
  - Pig
  - Hive
  - ...

Two strong statements about Hadoop

Use it

- If your primary use case is processing data that can be represented in flat files
- AND that data volume is at least in the tens, maybe hundreds of TB*
  - or there's a reasonable chance that it might be, within your app's expected lifespan
- AND your deployment environment is the cloud or you can dedicate > 4 servers,
- use it.

Don’t use it

- if any of those are not true.
  - Especially if your primary use case isn’t processing data.
    - If you can fit your data in Datomic (or something like it), you probably should.
    - If you can fit all of your data in memory across your cluster, you probably should.

Clojure and Hadoop

- Clojure-Hadoop (Stuart Sierra, Alex Ott)
  - https://github.com/alexott/clojure-hadoop
- Cascalog (Nathan Marz)
  - https://github.com/nathanmarz/cascalog

More Hadoop

- Less good reasons not to use it:
  - need realtime results
  - algorithm isn't easy to express in MR
  - it's no longer the next big thing

- Alternatives:
  - Apache Drill (Google Dremel)
  - Apache Giraph (Google Pregel)
  - Spark

Requirements

- Cloud-hosted
  - Elastic MR or similar

    + ease of use

    + pay-as-you-go

    - flexibility (versions, configuration)

    - latency
- Self-hosted
  - >= 4 servers ("evaluation cluster")
  - an ops team that has done this before

Data processing: Questions?

Questions? Comments? Anecdotes?

- **2nd Scenario: Process lots of events**

Storm

- High-volume event processing framework
- Can be combined with Hadoop (Marz's "Lambda Architecture")

Storm: Use it

- You're starting a new project
- You're dealing primarily with events
- You're going to have a lot of them
- You can model your logic as a static network of processes that have events flow through them
  - Can have conditionals, joins, state...

Storm: Don't use it

- Not a fundamentally event-based problem domain
- You're not looking at Twitter-scale loads, and the problem would be trivial to model with queues

Requirements

- >= 3 machines for production
  - ZK, Nimbus, Worker(s)
  - more for HA
- Some flexibility with your classpath

Storm: Questions?

Questions? Comments? Anecdotes?

. 3rd Scenario: Work Queues

Work Queues

- Rich's preference

- “Just use a queue” is perhaps overly simplistic
  - Not rocket science
    - although NASA does use a work queue system
  - But easy to get wrong
  - Surprising complexity when you get into it
    - Resque is 2320 LOC (Ruby)
    - Celery is 21k LOC (Python)
    - even Swarmiji is >800 LOC (Clojure)

Work Queues, cont.

- This is the space that interests me the most
  - i find myself in it most
- not as sexy as data processing or events
- most distributed processing really falls into this space.
  - "I don’t have the facts to back this up"
  - Most people aren’t processing Twitter feeds in real time.

Work Queues, cont.

- No clear winner in the Java world
  - Python has Celery
  - Ruby has Resque, backgroundjob, delayedjob...
  - C, Perl, etc. libraries like beanstalkd

Options

- Distributed core.async[+] (coming soon?)
- JCSP *+
- Swarmiji
- die roboter*
- Quantum's task queue*
- Jesque
- Non-JVM options
  - Resque
  - beanstalkd
  - ...

+ raw material

*not used in production

Not queues

- Akka+
- Immutant's Pipelines+
- Infinispan's Distributed Executors
  - also Hazelcast, GridGain, ...

Coming soon...

Quantum Retail's entry into the distributed processing space

- Pluggable transports
- Configurable middleware
  - Pedestal-style interceptors?
- Flexible distribution algorithms (FIFO, work stealing, ...)
- Embraces core.async channels
- Open-sourced ASAP
  - was supposed to be by today, sorry... hung up on technicalities. Like naming.

Questions?

Questions? Comments? Anecdotes?