EVERYBODY WOULD LOVE TO SEE THEMSELVES LIKE THIS
OR AT LEAST LIKE THIS
BUT SOMETIMES ONE FEELS MORE LIKE THIS
DATA ON ITS WAY TO HISTORY
interrupted by analytics and silicon
IDEA

continuously run analytics on the presence
learn from the (recent) past
predict the near future
recognise patterns
detect anomalies
visualise
trigger actions
do all that (ideally) on one single machine
possibly hide scientific aspects from users
USE CASES

anomaly detection
fraud detection
situational pricing
product placement
inventory control / forecast
online bidding
semi-automated operations
...

MOTIVATIONS

decision speed
decision automation
ergy / power cost
operational cost
(curiosity / challenge)
PROCESSING (~100% ON EVENT STREAMS)

take in events

group

compute features  check hard facts

classify, predict  visualise, trigger

train
STATUS: WORK IN PROGRESS
LEARNINGS SO FAR, OR 3 THINGS THAT CHALLENGE
MATH
STRATEGIES FOR TAKING IN EVENTS

collect original payloads, limit, circulate
summarise streams
sample-as-you-go
discard
ALGORITHMIC CHALLENGES

- one-pass or hell-pass
- statistically significant sampling
- time-space trade-off
- summarisation
- numerically stable parallelism
Numerically Stable, Single-Pass, Parallel Statistics Algorithms

An Improved Data Stream Summary: The Count-Min Sketch and its Applications

Distributed Streams Algorithms for Sliding Windows*

Chapter 5
Little’s Law

Janine Bennett

Phillip B. Gibbons¹ and Srikanta Tirthapura²

¹ Intel Research Pittsburgh
Pittsburgh, PA 15213, USA
philip.b.gibbons@intel.com

² Department of Electrical Engineering
Iowa State University
Ames, IA 50010, USA
snt@iastate.edu

John D.C. Little and Stephen C. Graves
Massachusetts Institute of Technology
ALGORITHMIC CHALLENGES
PLATFORM CHOICES

C/C++
JVM
Erlang/OTP and C/C++
Node.js and C/C++
...

PLATFORM CHOICE

- C/C++
- JVM
- Erlang/OTP and C/C++
- Node.js and C/C++
JVM PROS

maturity
high market penetration
portability
JIT compilation
concurrency/parallelism abstractions
garbage collection
mechanical sympathy
functional programming
lots of reusable libs
JVM PROS, REVISITED

- maturity
- high market penetration
- portability
- JIT compilation
- concurrency/parallelism abstractions
- garbage collection
- mechanical sympathy
- functional programming
- lots of reusable libs
PORTABILITY

growing numbers and computation speed lead to tighter dependency on OS and hardware and to partial nativeness.

Examples: thread affinity, GPUs/CUDA
We're building on Reactor, selectively glueing lock-free / single writer dispatchers (ring buffer) with parallel threads.

Java 8 is likely to change the game a bit
GARBAGE COLLECTION

Charles Nutter (@headius)
09.02.14 21:09
If you think GC is your problem, you're wrong. Your problem is allocation. Garbage isn't the disease...it's a symptom.
STRATEGIES TO KEEP AWAY GC PRESSURE

GC tuning
binary off-heap storage / payload referencing
off-heap data structures
preallocation and pooling
unsafe manipulation of primitives
PREMATURE OPTIMISATION IS EVIL
POST-HOC OPTIMISATION IS EVIL
THERE IS A WHOLE WORLD IN BETWEEN
MECHANICAL SYMPATHY

The bigger the numbers one wants to achieve, the less clean and beautiful the code will become, the more tricks will be coded directly, the higher the reliance on a specific JVM version, the higher the overall complexity.
FUNCTIONAL PROGRAMMING

FP is wonderful for data/event processing, streaming and math. We’re going with Clojure, but speed-critical parts are likely to be kept in Java.
Java ecosystem offers a lot of reusable libs, but in this area the options are rare. Some can be used, some can’t, some algorithms aren’t just implemented. We’re working on some implementations on our own, reusing where it makes sense.
ORTHOGONAL ASPECTS

time
speed-up of computations
testing
multi-machine
storage
network
**TIME**

In case an event carries a timestamp (logs), time is a natural data dimension.

Otherwise, this dimension needs to be applied artificially.

Time is enormously important on many levels.
STRATEGIES TO ARTIFICIALLY APPLY TIME DIMENSION

ad-hoc (slower)
dedicated thread (spinning, yielding, hashing wheel)
timer events vs. timestamp implantation
windowing by time
SPEED-UP OF COMPUTATIONS

GPUs are great for algorithm training and parallel computations (especially if it’s about matrices / vectorisation).

We’ve tested Rootbeer. Some of the challenges will be direct memory access and killing off-time computations.

FPGAs (maybe in the future)
Unit tests are hard on the level of complex topologies and time-dependent processing with nondeterministic event order.

We’re going with simple-check (QuickCheck-part-implementation in Clojure) for property testing.
MULTI-MACHINE

Ideally, it runs on one single machine. But at some point, this will be utopian.

It doesn’t have to be a fully blown distribution with horizontal gossip though.
MULTI-MACHINE STRATEGIES

multicast, first wins
anycast with replay / gap re-computation
logical / vertical split, merge
off-load algorithm training
STORAGE

Storage plays a secondary role - short term storage of original payloads for replay, long-term storage of computed features for offline training and exploration.

Approaches under consideration: 1-sec-BLOBs in HDFS, append-only-store with separate index
Network is extremely important when it’s about taking in events.

Log events are text, but growing numbers lead to binary, MTU-friendly events with careful packing.
LEARNING FROM / WITH

Mechanical sympathy people (LMAX, Real Logic etc.)
HFT people (OpenHFT etc.)
AddThis people (streaming, summarising etc.)
Netty people (off-heap, time etc.)
Reactor people (streaming on the JVM)
Cassandra people (algorithms in general)
ML people (time-efficient algorithms)
OpenJDK people (jol etc.)
CEP people
Many other people in the corresponding areas
With quite a complex graph of computations, we are trying to achieve a stable million events per second on a conventional server and go on from there.

Taking in events and actually processing them are different numerical views. It heavily depends on the requirements, payload format and overall processing complexity.
- pavlo.baron@codecentric.de
- @pavlobaron
THANK YOU
movie images taken everywhere on the internet and direct or indirect property of the movie companies.