CONVIVA® THE PULSE OF ONLINE VIDEO

Using Spark @ Conviva

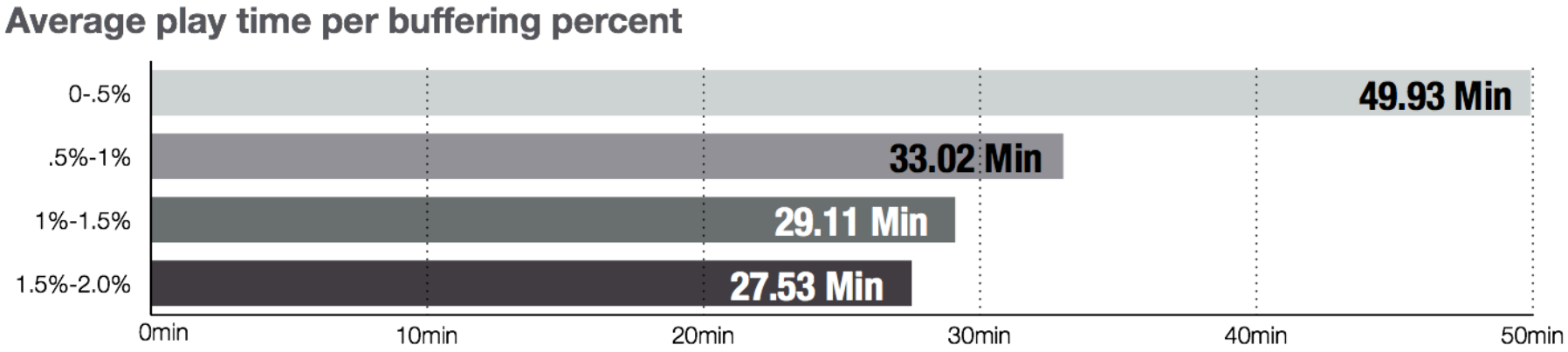
Spark Summit 2013

Summary

- Who are we?
- What is the problem we needed to solve?
- How was Spark essential to the solution?
- What can Spark enable us to do in the future?

Some context…

- Quality has a huge impact on user engagement in online video

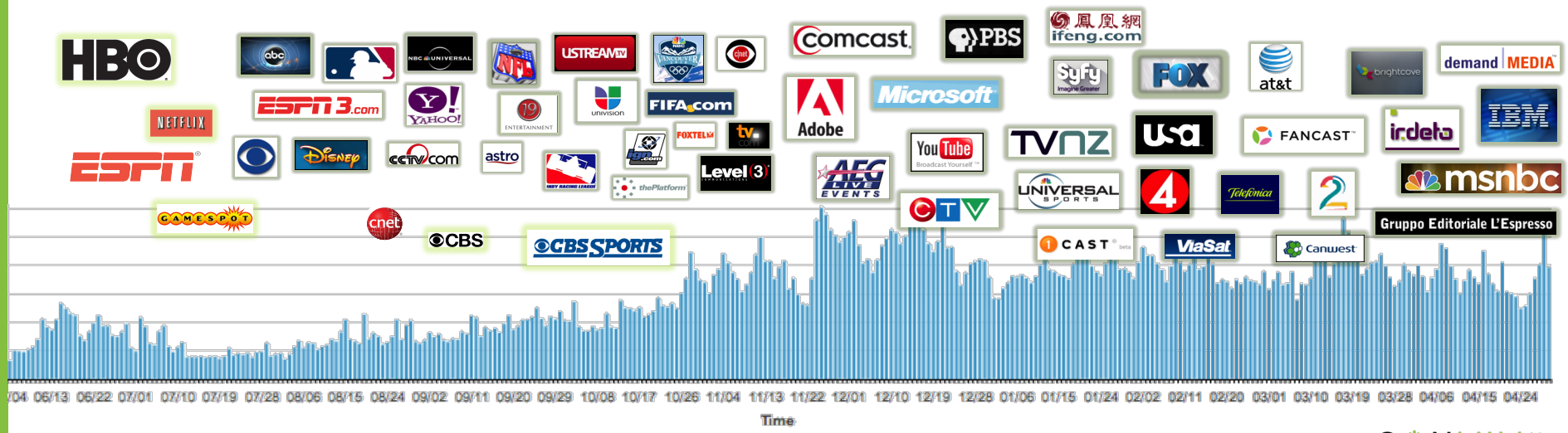


What we do

- Optimize end user online video experience
- Provide historical and real-time metrics for online video providers and distributers
- Maximize quality through
  - Adaptive Bit Rate (ABR)
  - Content Distribution Network (CDN) switching
- Enable Multi-CDN infrastructure with fine grained traffic shaping control

What traffic do we see?

- About four billion streams per month
- Mostly **premium** content providers (e.g., HBO, ESPN, BSkyB, CNTV) but also **User Generated Video sites** (e.g., Ustream)
- **Live events** (e.g., NCAA March Madness, FIFA World Cup, MLB), **short VoDs** (e.g., MSNBC), and **long VoDs** (e.g., HBO, Hulu)
- Various streaming protocols (e.g., Flash, SmoothStreaming, HLS, HDS), and devices (e.g., PC, iOS devices, Roku, XBOX, …)
- Traffic from all major CDNs, including ISP CDNs (e.g., Verizon, AT&T)

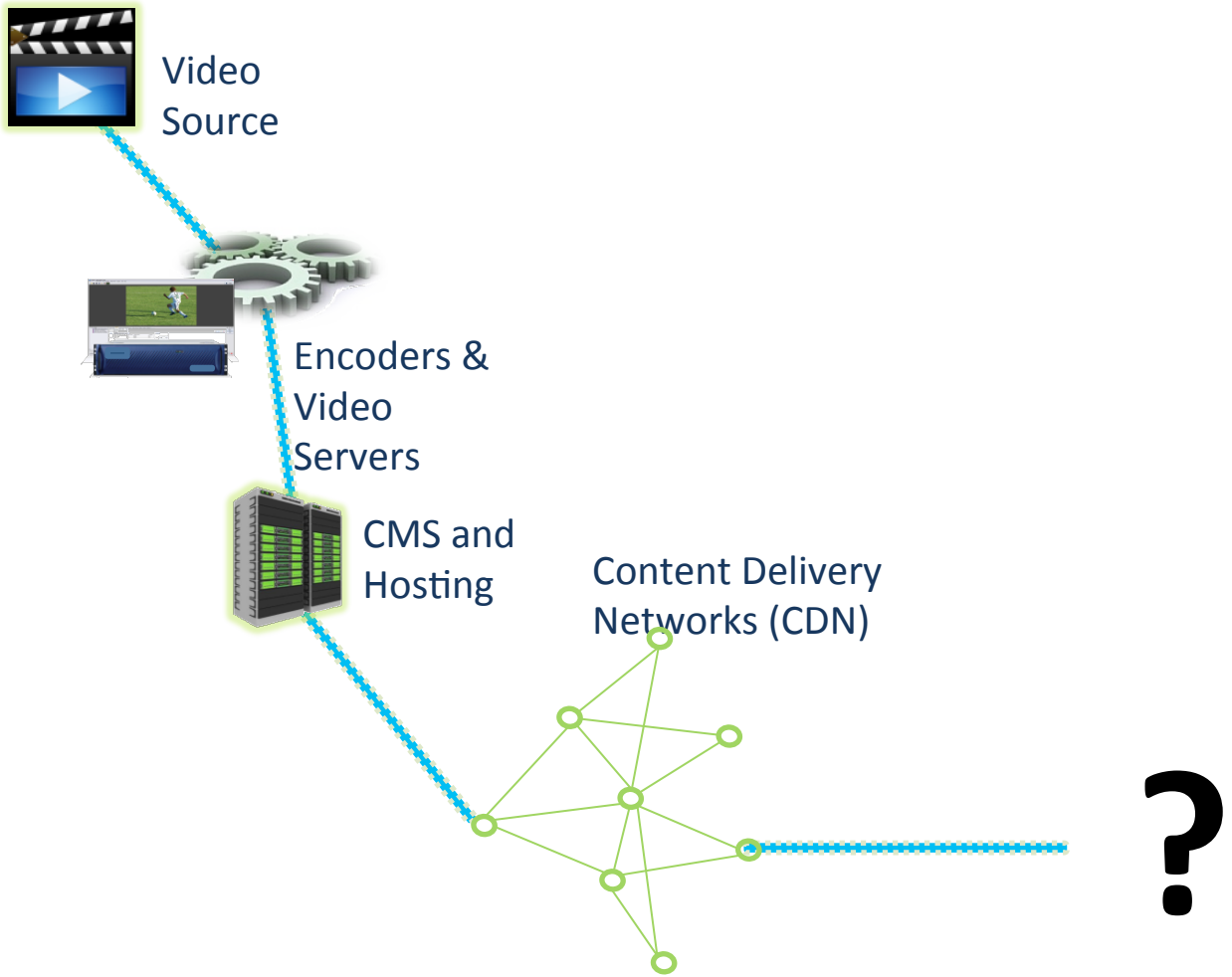


The Truth

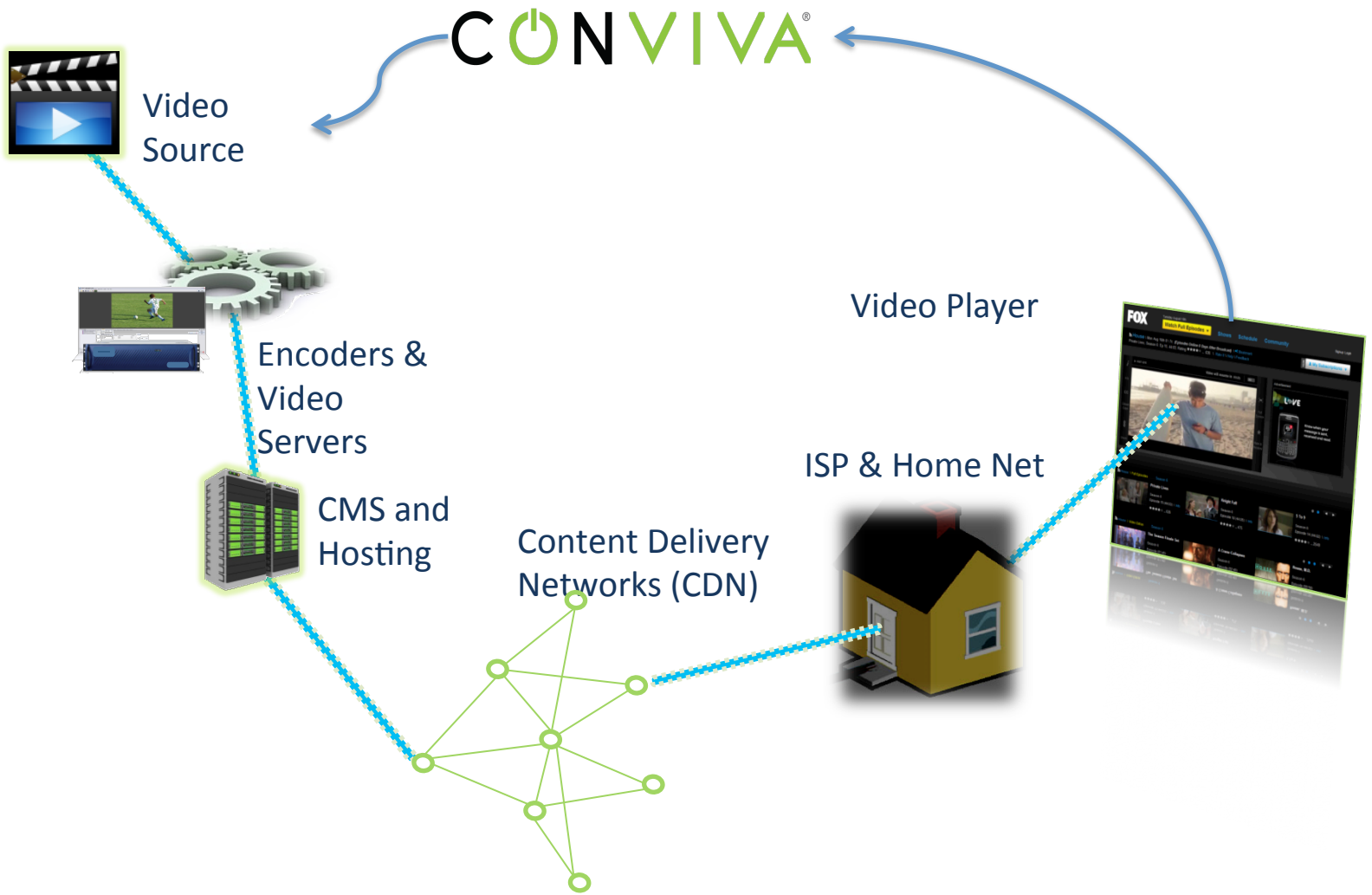
- Video delivery over the internet is hard
  - There is a big disconnect between viewers and publishers

The Viewer's Perspective

The Provider's Perspective

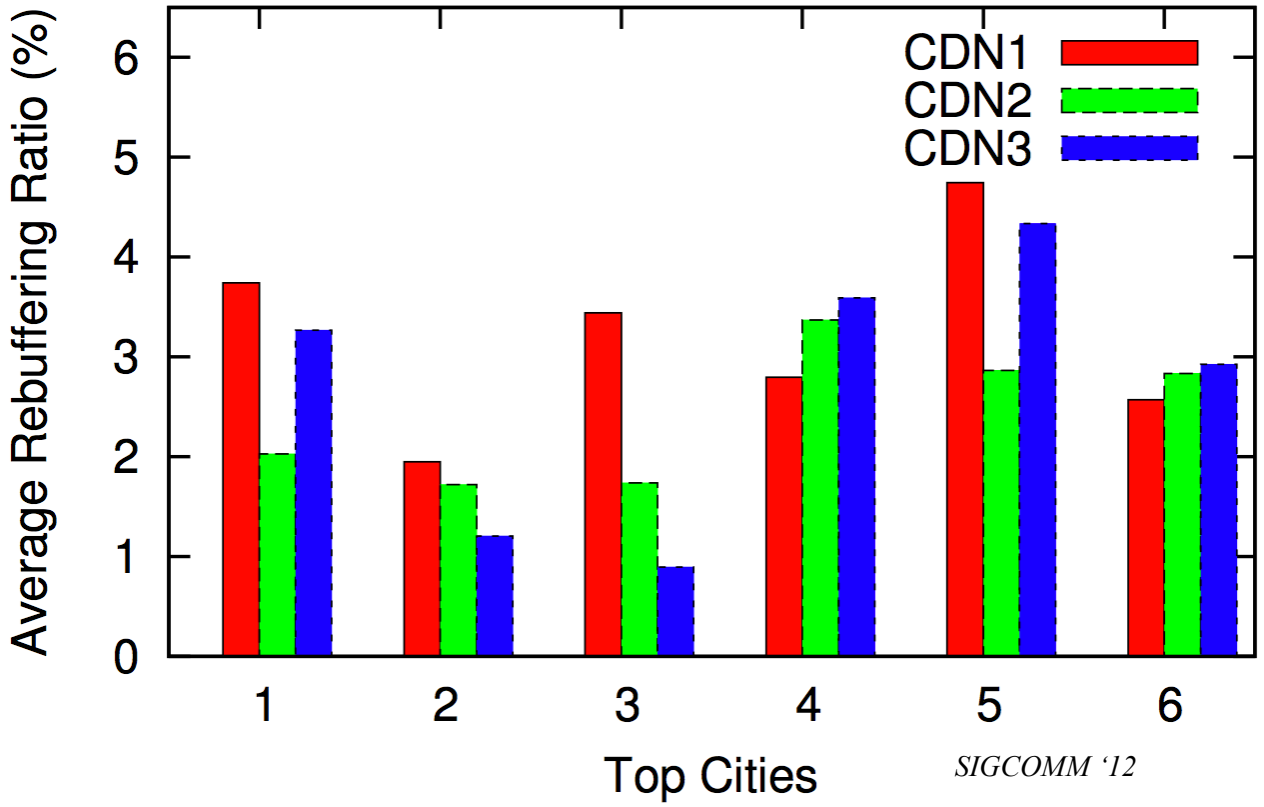


Closing the loop



The Truth

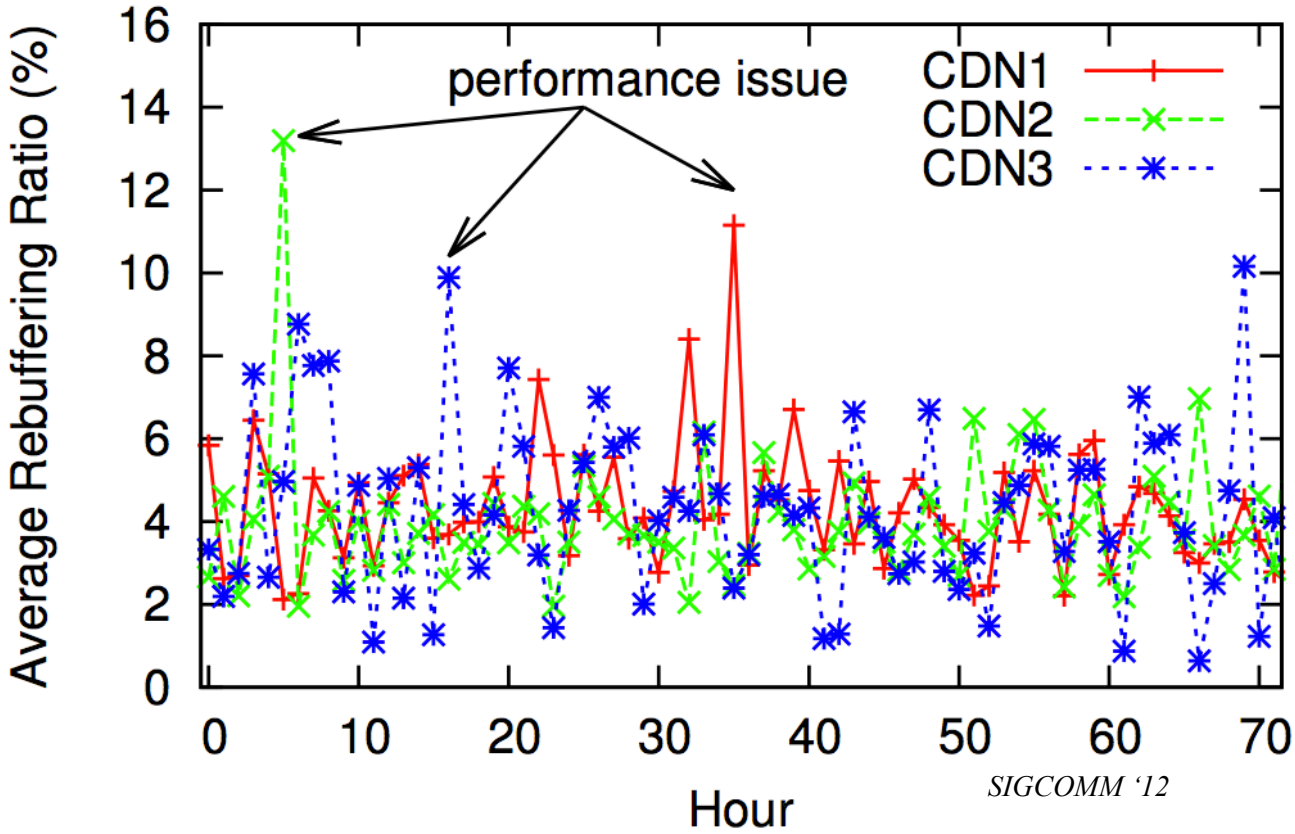
- Video delivery over the internet is hard
  - CDN variability makes it nearly impossible to deliver high quality everywhere with just one CDN



SIGCOMM '12

The Truth

- Video delivery over the internet is hard
  - CDN variability makes it nearly impossible to deliver high quality all the time with just one CDN



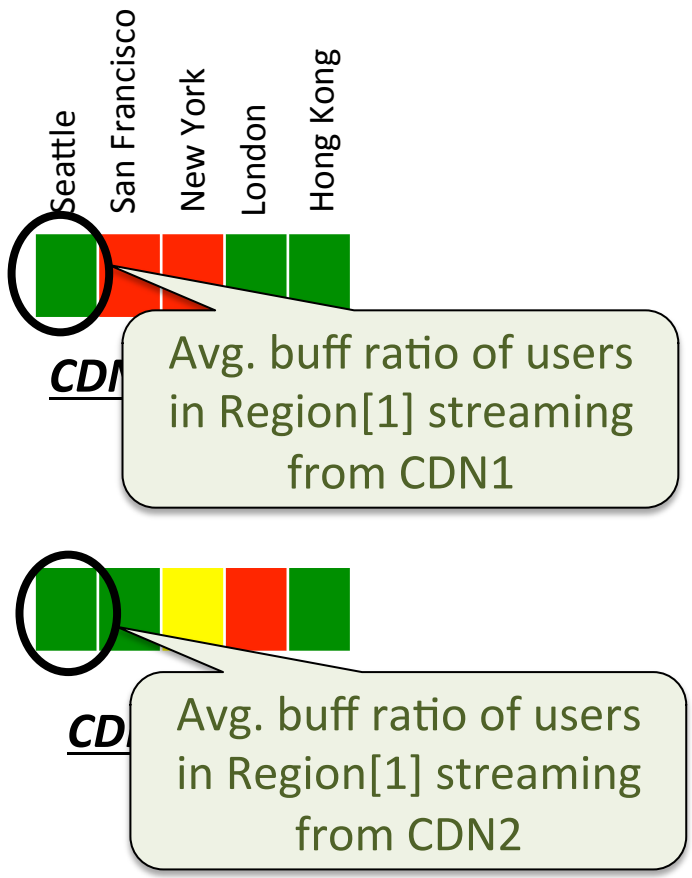
SIGCOMM '12

The Idea

- Where there is heterogeneity, there is room for optimization
- For each viewer we want to decide what CDN to stream from
- But it's difficult to model the internet, and things can rapidly change over time
- So we will make this decision based on the real-time data that we collect

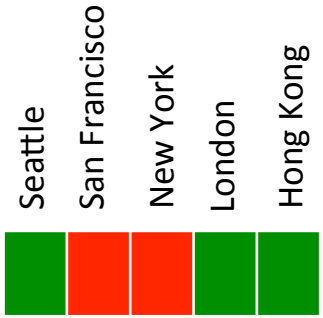
The Idea

- For each CDN, partition clients by City
- For each partition compute Buffering Ratio



The Idea

- For each partition select best CDN and send clients to this CDN

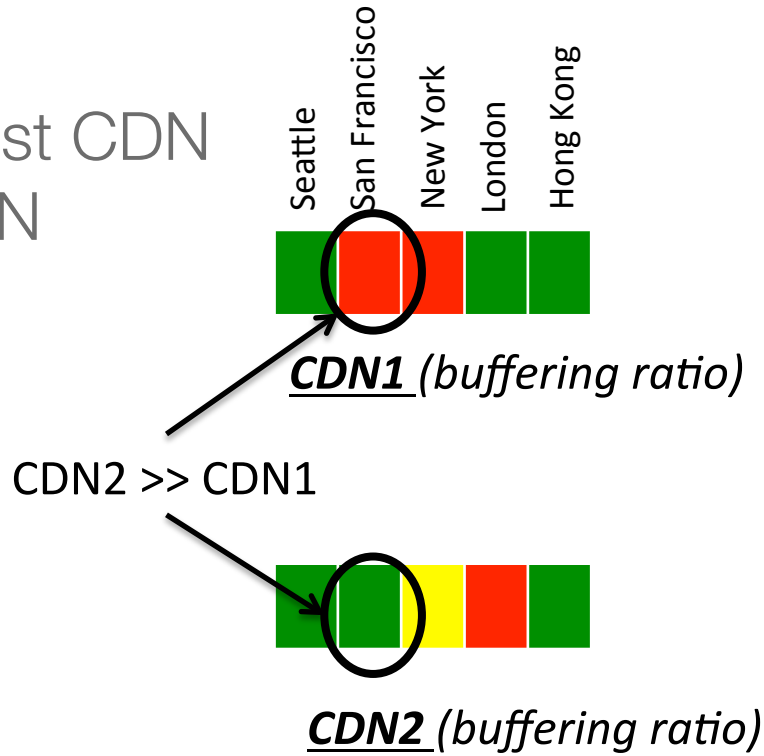


CDN1 *(buffering ratio)*

CDN2 *(buffering ratio)*

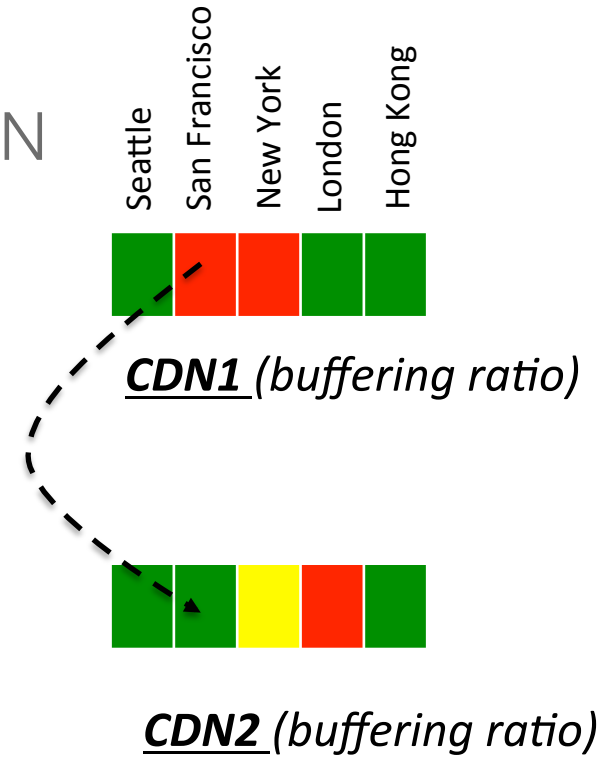
The Idea

- For each partition select best CDN and send clients to this CDN



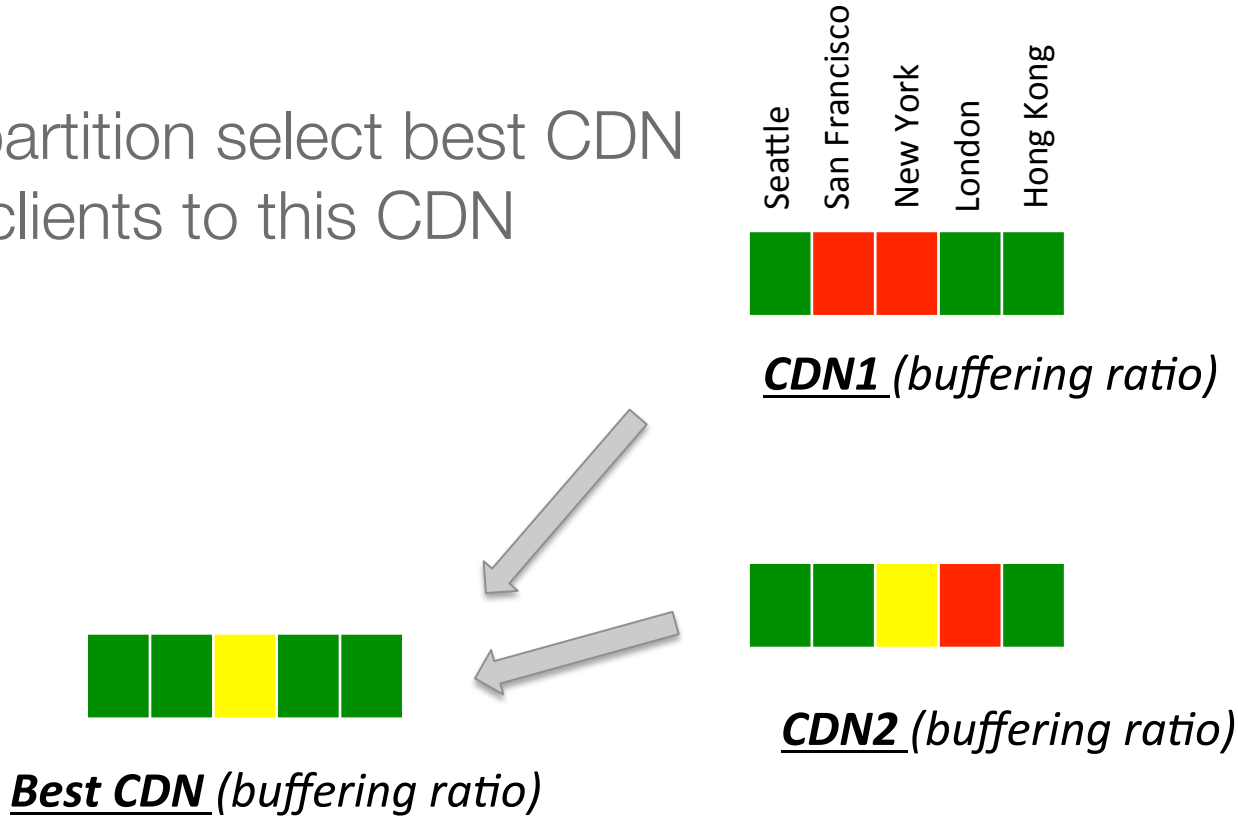
The Idea

- For each partition select best CDN and send clients to this CDN



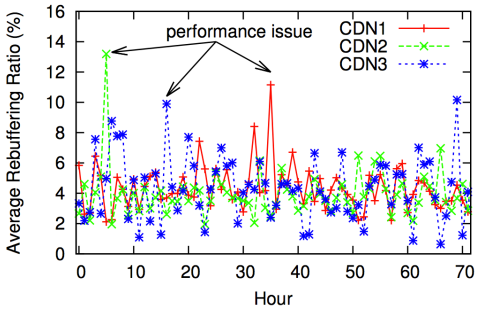
The Idea

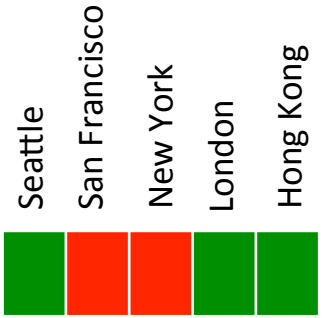
- For each partition select best CDN and send clients to this CDN



The Idea

- What if there are changes in performance?





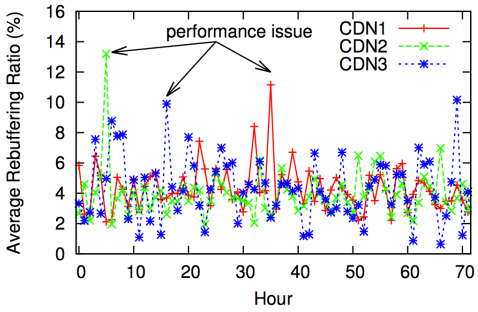
CDN1 *(buffering ratio)*

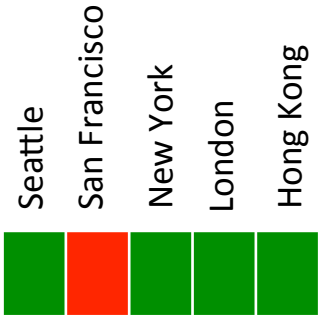
CDN2 *(buffering ratio)*

Best CDN *(buffering ratio)*

The Idea

- Use online algorithm respond to changes in the network.





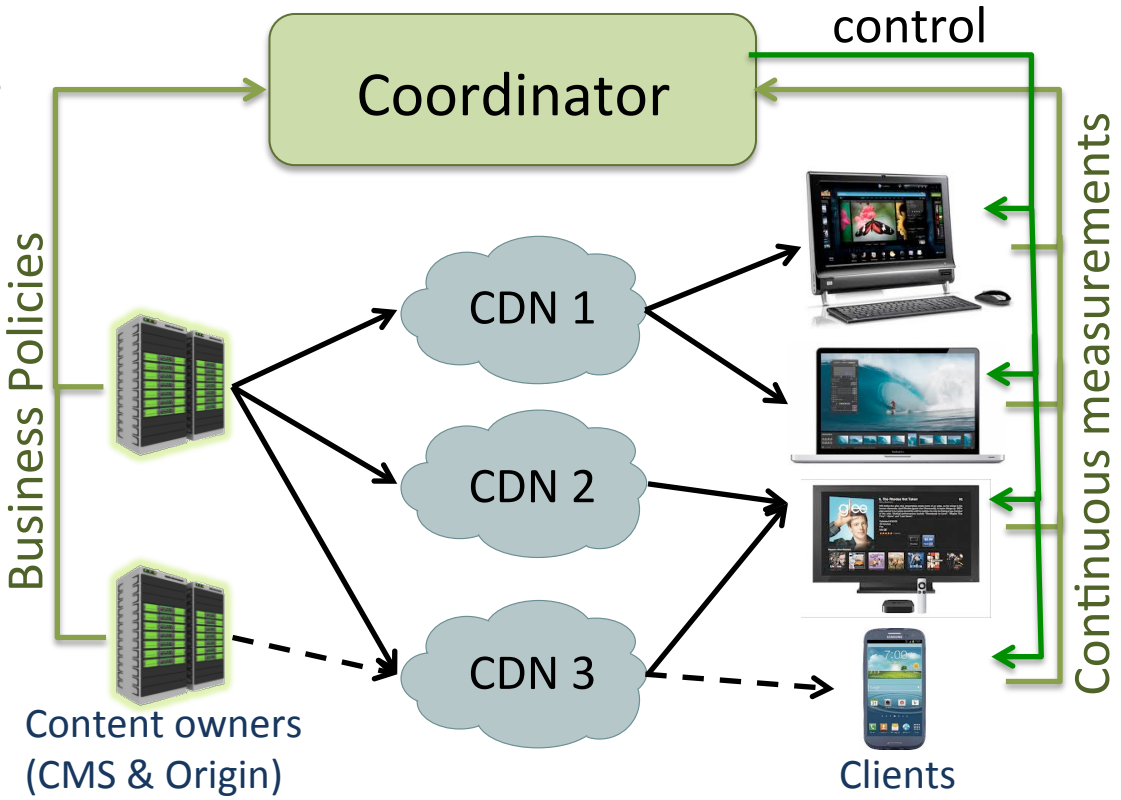
CDN1 *(buffering ratio)*

CDN2 *(buffering ratio)*

Best CDN *(buffering ratio)*

How?

- Coordinator implementing an optimization algorithm that dynamically selects a CDN for each client based on
  - Individual client
  - Aggregate statistics
  - Content owner policies
- All based on real-time data

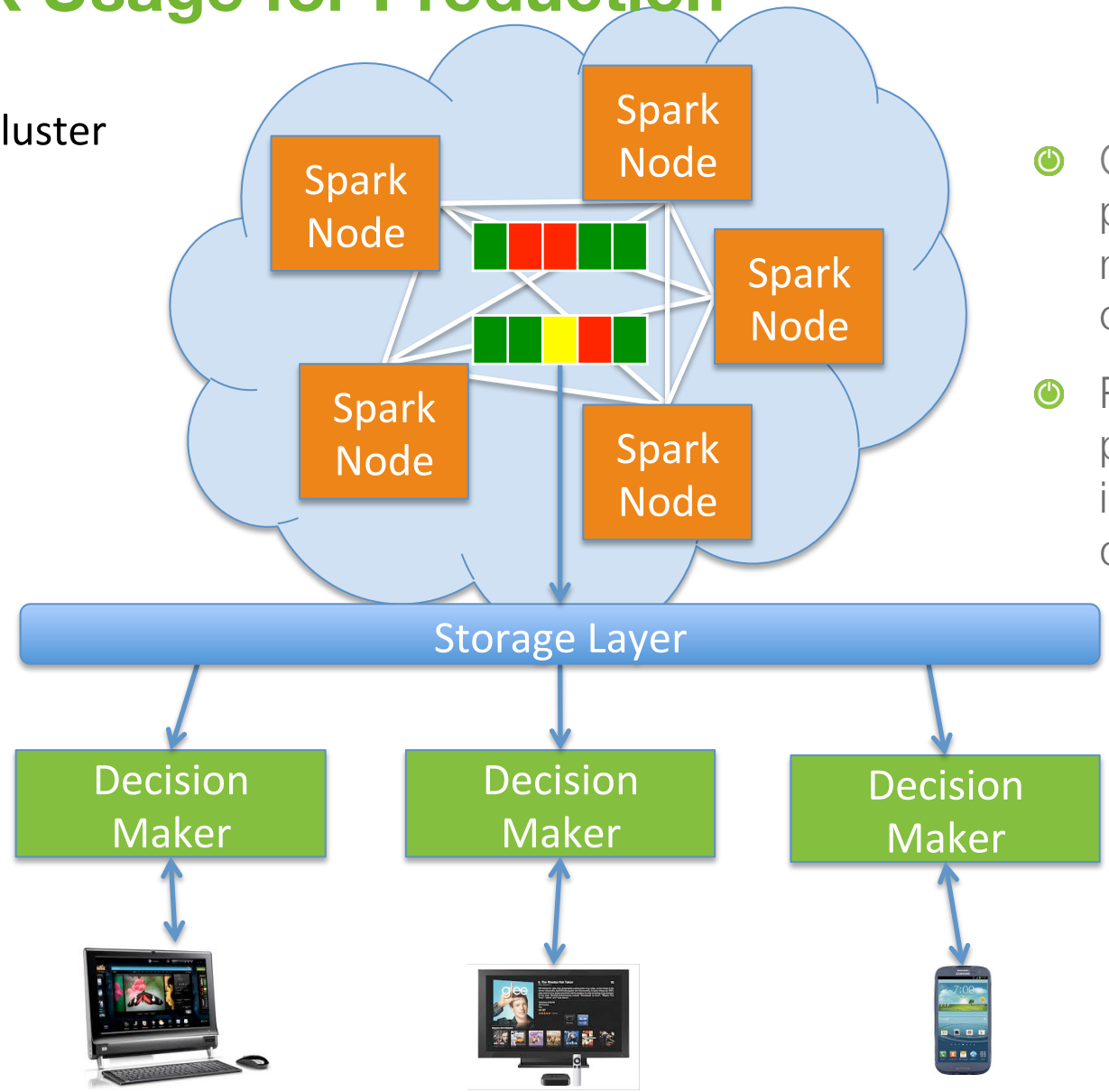


What processing framework do we use?

- Twitter Storm
  - Fault tolerance model affects data accuracy
  - Non-deterministic streaming model
- Roll our own
  - Too complex
  - No need to reinvent the wheel
- Spark
  - Easily integrates with existing Hadoop architecture
  - Flexible, simple data model
  - Writing map() is generally easier than writing update()

Spark Usage for Production

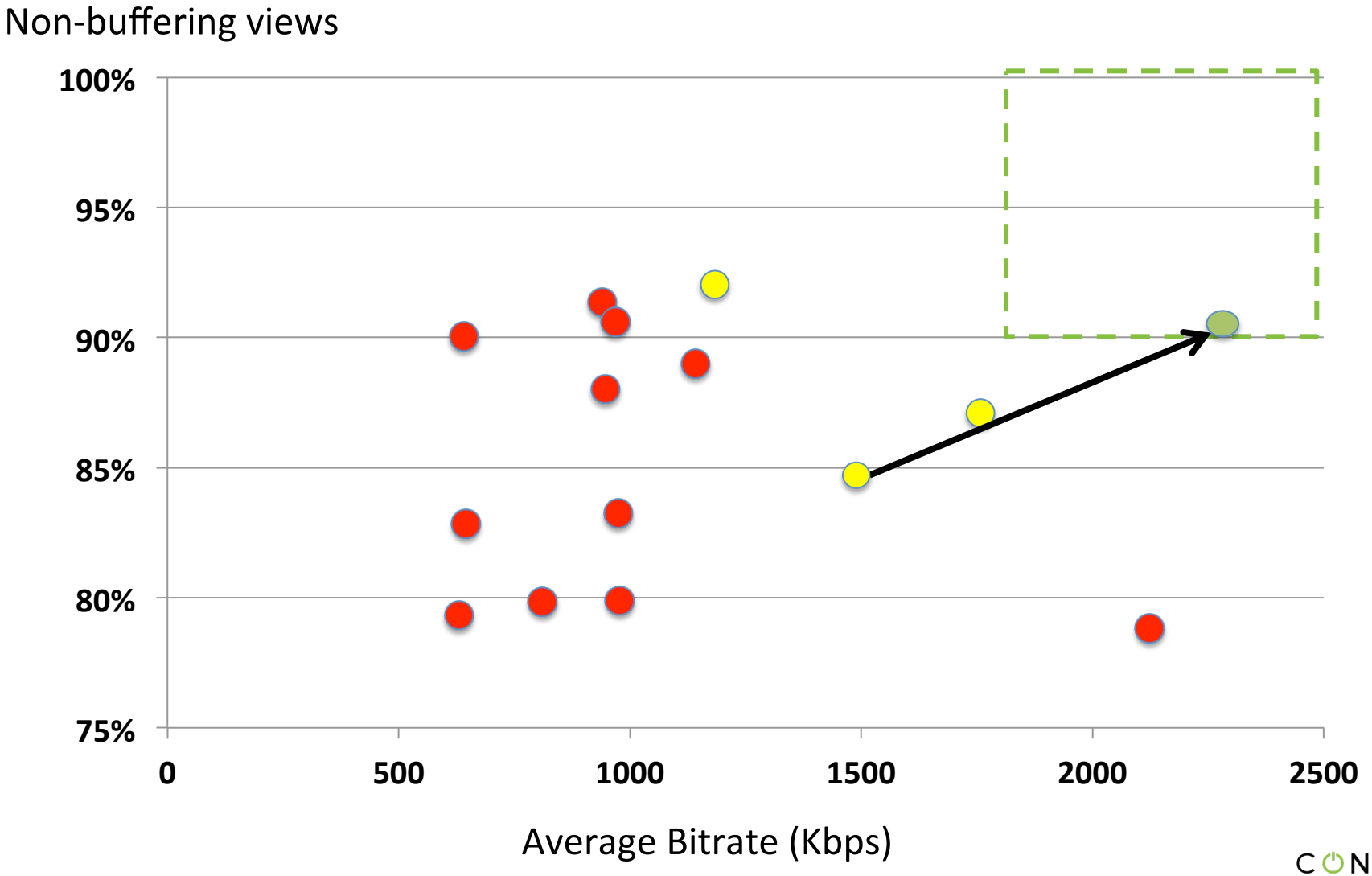
Spark Cluster



- Compute performance metrics in spark cluster
- Relay performance information to decision makers

Clients

Results


Non-buffering views
100%
95%
90%
85%
80%
75%
0
500
1000
1500
2000
2500
Average Bitrate (Kbps)

Spark's Role

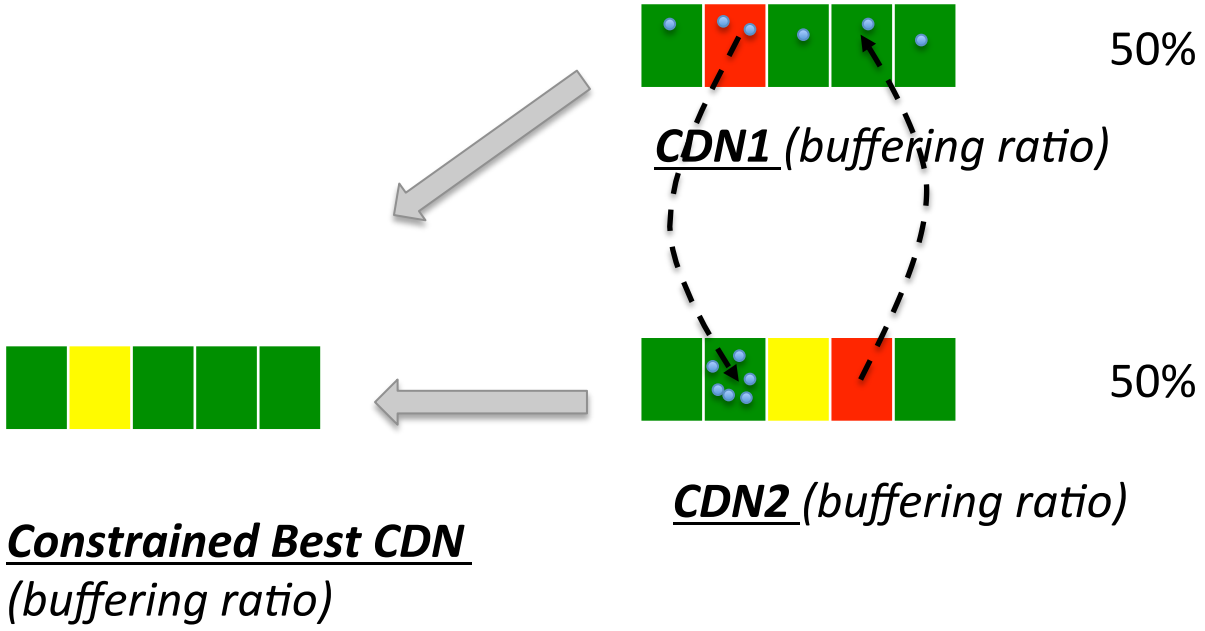
- Spark development was incredibly rapid, aided both by its excellent programming interface and highly active community
- Expressive:
  - Develop complex on-line ML decision based algorithm in ~1000 lines of code
  - Easy to prototype various algorithms
- It has made scalability a far more manageable problem
- After initial teething problems, we have been running Spark in a production environment reliably for several months.

Problems we faced

- Silent crashes…
- Often difficult to debug, requiring some tribal knowledge
- Difficult configuration parameters, with sometimes inexplicable results
- Fundamental understanding of underlying data model was essential to writing effective, stable spark programs

Enforcing constraints on optimization

- Imagine swapping clients until an optimal solution is reached



Enforcing constraints on top of optimization

- Solution is found *after* clients have already joined.
- Therefore we need to parameterize solution to clients already seen for online use.
- Need to compute an LP on real time data
- Spark Supported it
  - 20 LPs
  - Each with 4000 decisions variables and 350 constraints
  - 5 seconds.

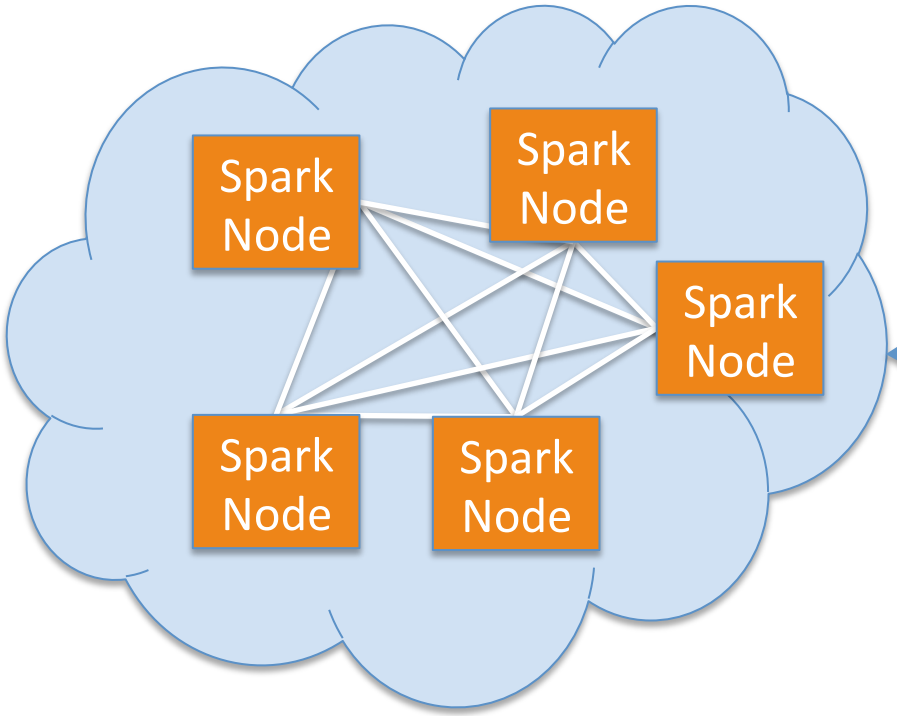
Tuning

- Can’t select a CDN based solely on one metric.
  - Select utility functions that best predict engagement
- Confidence in a decision, or evaluation will depend on how much data we have collected
  - Need to tune time window
  - Select different attributes for separation

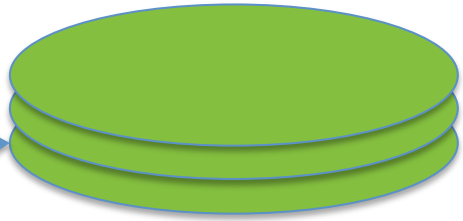
Tuning

- Need to validate algorithm changes quickly
- Simulation of algorithm offline, is essential

Spark Usage for Simulation



HDFS with Production traces



- Load production traces with randomized initial decisions
- Generate decision table (with artificial delay)
- Produce simulated decision set
- Evaluate decisions against actual traces to estimate expected quality improvement

In Summary

- Spark was able to support our initial requirement of fast fault tolerant performance computation for an on-line decision maker
- New complexities like LP calculation ‘just worked’ in the existing architecture
- Spark has become an essential tool in our software stack

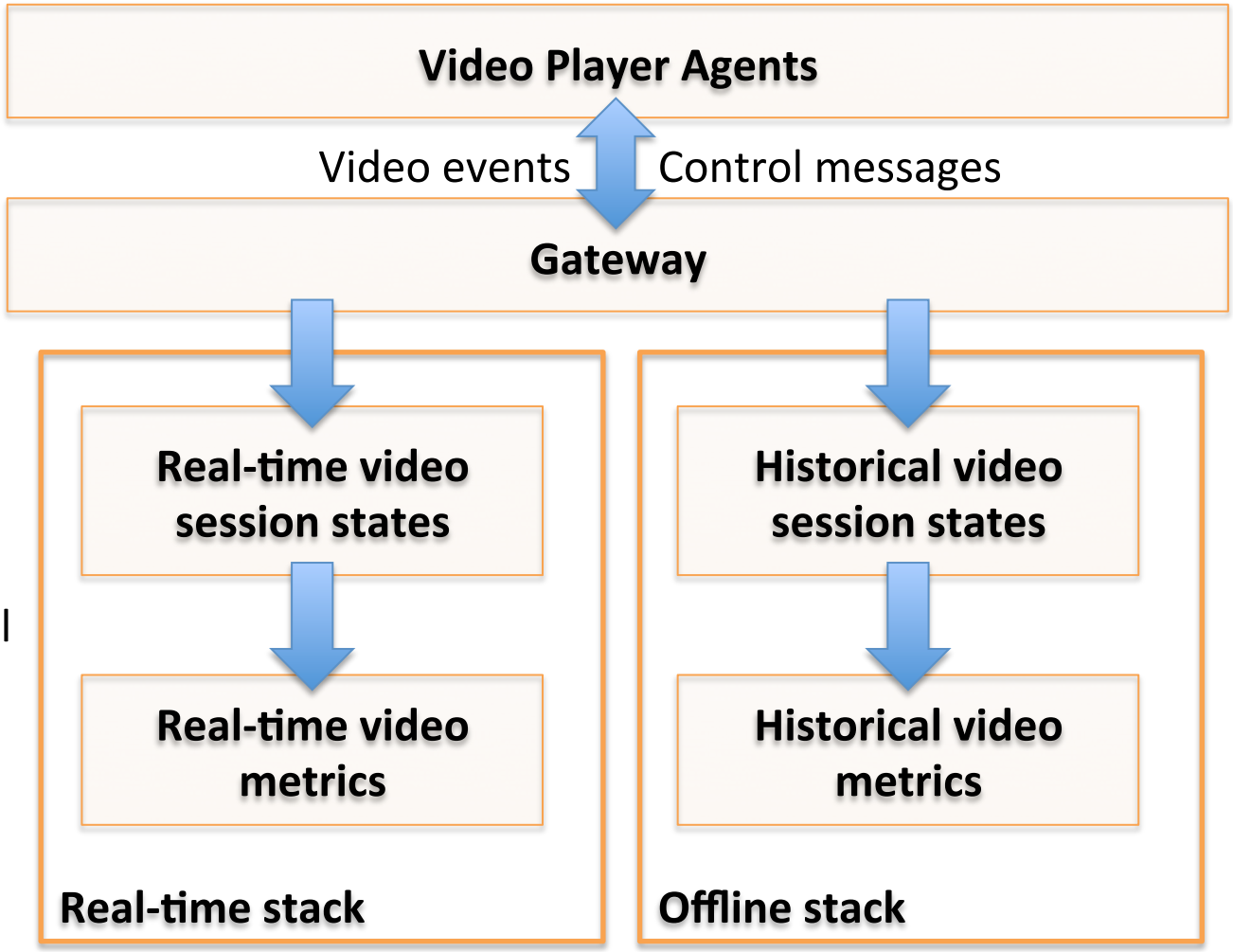
Thank you!

Questions?

What's Next?

- Our pilot Spark application is a success, so what's next?
- Many Spark applications in our roadmap
- Unification of real-time, near-real-time, and offline stacks using Spark and Streaming Spark

Current Dual-Stack Architecture



1-2 seconds

ephemeral

In-house streaming system

5min, hourly, daily, monthly

persistent

Run on Hadoop

It looks good, except

- The devil is in the details

What Cause Us Headaches?

- Streaming model is complicated
- Harder to reason about than batch model: cause more bugs
- Non-deterministic: harder to debug

- Maintain two code bases is costly
- Write code twice
- Out-of-sync implementation causes discrepancy

Streaming Spark to The Rescue

- Streaming Spark is a "streaming" system with a batch model under the hood
- Short summary: Spark and Streaming Spark provide *unified batch model, unified API, and unified fault tolerance model*
- For application, write code once, and run in both real-time and offline mode with different batch sizes
- Can provide exactly-once semantics

Lines of code for custom stack	Lines of code for unified stack
18K real time stack, 10K Hadoop application code, 10k shared code, all in Java	5K shared Spark application code in Scala + Java

Every Design Decision Has A Tradeoff

- Streaming Spark does a good job optimizing for small batches
- However, we still see a 2x performance degradation.

Real-time Stack Performance for custom stack	Real-time Stack Performance for unified stack
200K concurrent sessions per c3.2xlarge node	100K concurrent session per c3.2xlarge node

- More work needed on API, e.g., update config file for every batch
- Better input fault tolerance especially if input is from Kafka
- Dynamic batch size depending on load

An Alternative Solution From Twitter

- Twitter Storm, Trident, and SummingBird
- Storm is the popular streaming framework
- Trident provides batch processing and exactly-once semantics
- SummingBird provides write-code-once capability

- Streaming Spark is designed to do batch processing from scratch.
- Twitter systems are probably more battle-tested

Conclusion

- Simplicity rules and that's why we like the simple batch model
- Streaming Spark and Spark enable us to unify real-time, near-real-time, and offline stacks under the simple batch model
- Extra resource are required, but can be justified by lower code maintenance cost and faster dev iterations

Thanks!

- We have many projects planned on Spark platform.
- Stop by our booth if you are interested.

Questions?