Welcome!

To the 3rd Community Over Code Performance Engineering Track

Track chairs: Paul Brebner, Roger Abelenda
1st New Orleans 2022
Technologies: Lucene, Ozone, Kafka, Cassandra, JMeter, and Spark/ML
2nd Beijing 2023
Technologies: Kafka, JMeter, Arrow, Java profiling, Spark & Flink, Hadoop
3rd Halifax 2023
Technologies: Kafka, Ozone, Cassandra, Camel, JMeter/Selenium, Lucene

And maybe 4th EU 2024
Talks (3 x 2 = 6)

11:20 Paul Brebner
Developing Fast Applications With Open Source Software - Without The Fury (Kafka)

12:10 Duong Nguyen, Tanvi Penumudy, Ritesh Shukla
Design patterns and then the road to realize billions of objects, and exabytes of capacity, while preserving performance in Apache Ozone

--- LUNCH

2:20 German Eichberger, Pallavi Iyengar
Performance measurement and tuning of Cassandra 5.0 transactions on Cloud infrastructure

3:10 Otavio Piske
Hunting Performance Monsters on the Back of a Camel

--- COFFEE

4:10 Roger Abelenda
Quick load testing from Selenium scripts

5:00 Stefan Vodita
Lessons Learned from Benchmarking Amazon’s E-commerce Search Engine
Fast Open Source Software — Without The Fury

Paul Brebner - Open Source Technology Evangelist
www.instaclustr.com/paul-brebner/

Community Over Code Halifax October 2023
Fast *Cars*?!  
“NetApp helps Aston Martin F1 use data to go faster”

Massive amounts of data is captured and used to gain milliseconds of performance improvements on race-days

Using NetApp Cloud and Storage Technologies
Fast *Cars*?! “Fast & Furious”

“Fast & Furious” Cars for my Grandson

(unopened as they are “collectables”)

[Image of Fast & Furious Hot Wheels cars]
Fast Open Source Software

(Source: Wikimedia.org)
Enabled by Scalable Big Data Technologies (e.g. Apache Kafka®)

(Source: wikimedia.org)
And Cloud Infrastructure

Cars run on roads
S/W runs on H/W

(Source: wikimedia.org)
Without the Fury

Fury 1  Too Many Kafka Topics
Fury 2  Slow Consumers
Fury 3  Too Many Kafka Partitions
Fury 4  Single Threaded Consumers—concurrency limited by partitions
Fury 5  Operational Problems

(Source: Shutterstock)
Instaclustr Managed Platform

Cloud Platform for Big Data
Open Source Technologies

Focus of this talk is on
Apache Kafka®
Kafka is a distributed streams processing system—it allows distributed producers to send messages to distributed consumers via a Kafka cluster.
Partitions Enable Concurrency: Cluster and Producers

Kafka cluster
6 brokers replication 3
1 topic 4 partitions

Producers
Send messages to topic

M1 sent to leader of partition 0
Broker 1

Topic
M1
M2
Topic is replicated over multiple partitions and brokers

Broker 2
Broker 3

Partition 0
Leader
Partition 0
Follower
Partition 1
Follower
Partition 1
Leader
Partition 1
Follower
Partition 2
Follower
Partition 2
Leader
Partition 2
Follower
Partition 3
Follower
Partition 3
Leader
Partition 3
Follower

Kafka write scalability:
Multiple messages can be written concurrently using multiple partitions on different brokers.

Concurrent writes
To leaders on brokers 1 and 6 and followers on brokers 2, 3, 4, 5

M2 concurrently sent to leader of partition 2 on another broker

© Instaclustr Pty Limited, 2023
Partitions Enable Concurrency: Cluster and Consumers

Kafka cluster
6 brokers replication 3
1 topic 4 partitions

Read messages from topic

Kafka read scalability:
Multiple messages can be read concurrently
From leaders of multiple partitions and brokers (and followers)

Concurrent reads
From Brokers 1, 2, 5, 6

© Instaclustr Pty Limited, 2023
Partitions enable Consumers to share work (c.f. Amish Barn raising) within a consumer group.
Multiple Groups Enable Message Broadcasting

Messages are duplicated (c.f. clones) across groups, as each consumer group receives a copy of each message.
Fury 1: Too Many Kafka Topics

“Kongo” Logistics IoT Application
1. **Many (100s) of Topics**

- 100s of locations (Warehouses, Trucks)
- Each location has a topic and multiple consumer groups (so all the Goods in a location receive relevant events)

**Option 1:**
- Many topics
- Many consumer groups per topic -> high fan-out
2. One Topic, One Consumer Group

- One topic for all locations
- Using an external notification mechanism for event broadcasting (Guava Event Bus)
Single Topic/Single Consumer Group Wins

155 times better! But why?

Many topics, many consumer groups, 7200

Single topic, single consumer group, 1120000

© Instaclustr Pty Limited, 2023
Trains Are More Scalable Than Cars

Train in the Canadian Rockies (Source: Getty Images)
High Fan-Out = Lots of On/Off Ramps

Explanation: High fan-out = lots of output data and many consumer groups (resource intensive)

(Source: Shutterstock)
Many Topics = Traffic Jam

Explanation:
More topics → more partitions

(Source: Shutterstock)
But Kafka Is Scalable! Bigger Clusters

Vertical/Scale-up
Increase Node Sizes

Add more lanes!

(Source: Wikimedia)
Add more roads!
Fury 2: Slow Consumers
Kafka+Cassandra Anomaly Detector Application
Massively Scalable Anomaly Detection: Tuning Knobs (Orange h/w, Yellow s/w)

- Initially just increased h/w resources
- Scaling was “easy” with Kubernetes
- Easy to create lots of consumers (100s)
- Initially single threaded Kafka consumer (no thread pools)
But Scalability Not Great: Scaling Is Too Easy—Scalability Harder
Default Kafka consumers:

- Are single-threaded
- If the processing is “slow” then queuing occurs—as the thread is blocked—reducing throughput
- Solutions include speed up the processing, or increase the number of consumers
- But more consumers → more partitions
- As each consumer needs 1 or more partitions

(Source: Getty Images)
Single Threaded Kafka Consumers And Slow Processing = Slow Consumers

- Slow consumers → need more consumers and also more partitions for higher throughput
- But more consumers is slow, try speeding them up
We Need Some Car Mods (Hacks)
Multi-Threaded/Two Pool Consumers

The famous Bondi Ocean Pool (in Sydney, Australia) has 2 pools

(Source: Shutterstock)
Tuning: Optimize Consumer Speed/Concurrency Using 2 Stage Pipeline

Less consumers (around 100) gives higher throughput—a surprise!

1. Speed up polling (thread pool 1)
2. Maximize anomaly detector concurrency (thread pool 2)

Result—Reduces the number of consumers and therefore partitions needed and gives higher throughput—why?

Don’t more partitions give higher throughput?! Answer in part 3.
Scalability Post-Tuning—7.5 to 19 Billion Checks/Day—2.5 Times Improvement

Total Cores vs. Billions of Checks/Day (pre-tuning)

Billions of checks/day (pre-tuning) vs. Billions of checks/day (post-tuning)
Fury 3: Too Many Partitions
What’s really going on under the Kafka Bonnet?

(Source: Adobe Stock)
Partitions = Pistons (Cylinders)

(Source: Getty Images)
But How Many?

1 piston isn’t very powerful:
Isetta (bubble car)
single-cylinder, 10HP,
top speed 55MPH

Isetta 1-cylinder car
(Source: Wikimedia)
...16 Pistons Is a Lot!

Cadillac V-16
175HP,
100 MPH!
Can You Have “Too Many”?  

YES!
Experimental 42-cylinder 2,350 hp (plane) engine!

Source: Wikimedia
You need sufficient partitions to benefit from the cluster concurrency—And not too many that the replication overhead impacts overall throughput.
2022 Kraft/Zookeeper Modes vs. 2020 Results Better, 1000 Partitions Is Ok

You need sufficient partitions to benefit from the cluster concurrency and not too many that the replication overhead impacts overall throughput.

© Instaclustr Pty Limited, 2023
Fury 4: Single Threaded Consumers

What’s New? Kafka Parallel Consumer = A New Engine!

"La Jamais Contente", first car to reach 100 km/h in 1899 – 68hp electric!
(Source: Wikimedia)

Rimac Nevera – Electric “Hypercar”
4 Engines, 1,888 HP, 0-400KM/H in 29s
(Source: Wikimedia)
Theory: Little’s Law: Concurrency = Throughput x Time

Rearranged:
- Throughput = Concurrency/Time
- Concurrency is Partitions = Consumers
- Using default consumer the throughput drops with increasing time
- Only solution is to increase partitions
For Given Target Throughput (1M TPS)
Increasing Partitions With Increasing Time

You need sufficient partitions to benefit from the cluster concurrency and not too many that the replication overhead impacts overall throughput.
Order in Kafka Is Partition Based—So How To Increase Consumer Concurrency?

You need sufficient partitions to benefit from the cluster concurrency and not too many that the replication overhead impacts overall throughput.
Kafka Parallel Consumers: Multi-Threaded Consumer

- Multiple ordering options—c.f. default Kafka only guarantees order within partitions!

  PARTITION $\rightarrow$ KEY $\rightarrow$ UNORDERED

  Increasing concurrency $\rightarrow$

- Concurrency from 1 to lots—depends on client resources, and Partitions/Key space sizes

- KEY has higher concurrency than Partition and is ordered by KEY—reasonable compromise

- UNORDERED is unordered
Kafka Parallel Consumers
Multi-Threaded Consumer = Buses

You need sufficient partitions to benefit from the cluster concurrency
And not too many that the replication overhead impacts overall throughput

(Source: Getty Images)
Theoretical Improvement for Each Mode – max 3 orders of magnitude

You need sufficient partitions to benefit from the cluster concurrency, but not too many that the replication overhead impacts overall throughput. For 1,000 partitions and 100 consumers max: 

- (1) Default Consumer, RT=10ms
- (2) Parallel Consumer, RT=10ms, Partition Order
- (3) Parallel Consumer, RT=10ms, Key Order (1,000 keys)
- (4) Parallel Consumer, RT=10ms, Key Order (10,000 keys)
- (5) Parallel Consumer, RT=10ms, Key Order (100,000 keys)
- (6) Parallel Consumer, RT=10ms, Unordered
Experimental Results:
3, 50, and 200 times improvement, unordered best

You need sufficient partitions to benefit from the cluster concurrency And not too many that the replication overhead impacts overall throughput

Experimental Throughput Results, RT = 10ms
1 Consumer, 10 Partitions, 100 Keys

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Measured TPS</th>
<th>Theory TPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partition, 1 thread</td>
<td>89</td>
<td>100</td>
</tr>
<tr>
<td>Partition, 10 threads</td>
<td>300</td>
<td>1000</td>
</tr>
<tr>
<td>Key, 100 threads</td>
<td>4850</td>
<td>10000</td>
</tr>
<tr>
<td>Unordered, 200 threads</td>
<td>17465</td>
<td>20000</td>
</tr>
</tbody>
</table>

1 consumer
10 partitions
100 keys
10ms latency
Watch Out for the Kafka Furies

- Too many topics (= too many partitions)
- Too many consumer groups
- Slow consumers (= too many partitions)
- Insufficient/too many partitions
- Single threaded consumer (= too many partitions)
Speed can be achieved with train-buses

Track + Buses

Minimize Topics and Partitions = Tracks
- Buses are fast & self-driving on tracks

Minimize Consumer Groups = Interchanges
- At interchanges, buses fan out onto roads, reducing passenger transfers

Maximize Consumer Concurrency = Buses
- Multiple passengers, integrated with road system

Adelaide’s O-Bahn Busway train-bus system

Source: Wikimedia
Fury 5: Operational Problems

Pit Stop Performance Penalties (Source: Getty Images)
Even well designed Kafka Applications occasionally have operational performance problems.

Rapid increase in CPU Utilization from normal of 50% to lots

What’s going on?

Has the workload increased?
No.

Has the cluster capacity decreased? (e.g. lost some brokers)
No.

This was the only example of a Kafka performance problem in our Post-Incident Reviews.
Remediation

• Attempt 1: Replace 1 broker at a time
  • Didn’t work – problem reappeared when new broker took over partition leadership
  • (A broker restart is what triggered the problem)

• Attempt 2: Stop all customer clients (producers/consumers)
  • Perform a rolling restart of Kafka cluster
  • Restart clients, hold breathe...

Apollo 3rd stage J-2 Engines (13,000,000 HP) were designed to restart – but failed to restart in uncrewed Apollo 6
(Source: NASA)
Back to normal
Diagnosis: Kafka is a distributed system

Kafka clients are also a critical part of the system

Kafka Network Processor Threads handle client network data

Network Processor Idle % has decreased (more is better, less is worse)

The client load had increased.
Why? Further Clues and Causes

- Kernel Error
  - TCP: request_sock_TCP: Possible SYN Flooding ...

- Decreased Network Processor Idle % was a symptom
  - Of repeated Kafka producer connection attempts
  - TCP congestion control and window size dropped to very small and inconsistent values between producers and broker
  - Making it impossible to reconnect producers to brokers after a broker restart

- Cause?
  - Broker restart triggered the problem
  - Permanent fix required Linux Kernel and Kafka version upgrades
  - And different settings for SYN cookie options
  - Probably related to KAFKA issues 9648 and 764
And Back to the Start (Aston Martins)
Thank You and Goodbye (Eject)

“Ejector seat? You’re joking.”
My now “collectable” but played with Corgi DB5 (Source: Paul Brebner)
THANK YOU!