A year supporting Apache Kafka™

Dustin Cote (Customer Operations, Confluent)
Prerequisites

- Medium experience with Kafka
- Cursory knowledge of
  - Configuring Kafka
  - Replication
  - Request lifecycles
- Interest in Kafka Ops
- Don’t have these things?
  - Kafka: The Definitive Guide
Agenda

- Quick Flyover of Concepts
- Discussion on some techniques we use to generically troubleshoot
- Three things we’ve seen trouble with
  - For each one
    - What happened
    - Why it happened
    - What we’re doing to make it not happen again
- Wrap up and questions
Background

Why does it matter?
- Avoid the mistakes of others
- Reduce time to a stable production
- Help improve Kafka!

OK, but who should really care?
- **Admins** (what should I look out for/why should I upgrade?)
- **Developers** (how can I be a good citizen/why does my admin look at me like that?)
- **Architects** (what are good deployment strategies/how have we addressed problem use cases?)
Troubleshooting (Methodology) - USE

Summary:
Check Utilization, Saturation and Errors for each resource

Definitions:
- Utilization: How much work is being performed
- Saturation: No additional work can be performed
- Errors: Error, possibly Warn level messages in the logs

Reasoning:
- Avoid needless work
- Expedite TTR
- Accurate RCAs

Acknowledgments:
“Systems Performance: Enterprise and the Cloud”, Brendan Gregg
Monitoring (JMX)

- Why JMX?
  - Lightweight for the broker, lightweight for your storage
  - Designed for historical information and pattern recognition
  - Easily shared (could even publish them to Kafka!) and moved to a new (not local) device

- Critical metrics ([http://kafka.apache.org/documentation.html#monitoring](http://kafka.apache.org/documentation.html#monitoring))
  - Alert on these
  - **Alert != restart**

- How hard is it to set up?
  - Plenty of solutions of varying detail and price
  - Find what works for your org

- But what do all of these metrics mean??
Concept Overview: How Requests Flow in Kafka

- **Replication**: copying messages to other brokers for durability
- **ISR**: “In-sync Replica” -- is this replica up to date?
- Brokers both servers and clients
- Coherence matters
## Troubleshooting (JMX) - Utilization/Saturation

<table>
<thead>
<tr>
<th>Key</th>
<th>Resource utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Replica Manager</strong></td>
<td>UnderReplicatedPartitions</td>
</tr>
<tr>
<td></td>
<td>RequestHandlerAvgIdlePercent</td>
</tr>
<tr>
<td><strong>Request Handler Pool</strong></td>
<td>NetworkProcessorAvgIdlePercent</td>
</tr>
<tr>
<td></td>
<td>RequestQueueSize</td>
</tr>
<tr>
<td></td>
<td>RequestQueueTimeMs</td>
</tr>
<tr>
<td></td>
<td>LocalTimeMs</td>
</tr>
<tr>
<td></td>
<td>RemoteTimeMs</td>
</tr>
<tr>
<td></td>
<td>ResponseQueueSize</td>
</tr>
<tr>
<td><strong>Network Processor Threads</strong></td>
<td>RequestsPerSec</td>
</tr>
<tr>
<td></td>
<td>ResponseSendTimeMs,</td>
</tr>
</tbody>
</table>
Troubleshooting (Logging/Errors)

- Should not drive investigation
- Provides context to the observed metrics for further investigation
- Exceptions stacks are useful for spotting bugs
Example 1 -- ISR Shrink/Expand

- **Initial problem description**
  - Under-replicated partitions are growing

- **Scenario**
  - Issue self heals
  - NetworkHandlerAvgIdle stabilizes at 60%
  - Brokers are 0.10.0 with some 0.9.x clients
  - Kafkacat -L requests time out occasionally

- **Cause**
  - 0.9.x clients were slow to receive responses
  - A blocking call was used to send down converted messages to older clients
  - This tied up network processor threads
Example 1 -- ISR Shrink/Expand

● Prevention
  ○ Warn on ISR Shrinks/Expands
  ○ Warn on high Network and Request handler utilization/saturation
  ○ Be mindful of increasing request latency

● Solution
  ○ Upgrade to 0.10.0.1 with the permanent fix
  ○ Alternatively you could upgrade the clients

● Moral
  ○ Use the metrics available not assumptions to limit the scope of your investigation.
Example 2 – Failed automation

- **Initial problem description**
  - 1 broker goes “down” repeatedly
  - Full cluster restart, stabilizing for > 1 hour
  - After whole cluster is up, some partitions are permanently under-replicated

- **Scenario**
  - Environment: Cloud, Docker
  - For any failure, destroy/rebuild containers
  - Failure = ELB to broker connection failure

- **Cause**
  - Single broker lost connectivity with the ELB
  - Full cluster restart crushed the controller upon startup (8000+ partitions across 5 brokers).
  - Repeated automatic restarts during stabilization exacerbated problem
Example 2 - Failed automation
Example 2 -- Failed automation

- **Prevention**
  - Go to the source of truth for broker liveness, ZooKeeper
  - Alert and analyze upon “broker down” instead of triggering a container rebuild
  - Avoid “system reset” as a debugging tool

- **Solution**
  - Near term: disable controlled shutdown to avoid exposure
  - Long term: reduce the number of partitions and take preventative measures above

- **Moral**
  - Implement monitoring with JMX and rely on it
  - If you aren’t sure what action to take automatically, tell a human
  - Distributed systems and blind restarts do not mix
Example 3 – Reassignment Storm

- **Initial problem description**
  - Bad performance, producing is slow, consuming is slow, ISRs are shrinking

- **Scenario**
  - Adding a new broker
  - Partition reassignment done manually
  - Reassignment tool requires some knowledge of how replication works

- **Cause**
  - A cluster-wide partition reassignment was started
  - Brokers’ network processors overwhelmed
  - Crushed network processors == everything slows down
  - Prior to 0.10.1, process cannot be throttled
Example 3 - Reassignment Storm
Example 3 – Reassignment Storm

- **Prevention**
  - Take into account number of partitions being moved

- **Solution**
  - Move a small number of partitions at a time
  - Upgrade to 0.10.1 or higher to take advantage of replica throttling
    - [http://kafka.apache.org/documentation.html#rep-throttle](http://kafka.apache.org/documentation.html#rep-throttle)
  - Confluent Rebalancer

- **Moral**
  - Monitor the cluster with JMX to understand loading
  - Anytime you change how data is flowing, test in a stage environment if possible first
What did we learn...

- Implement monitoring with JMX and rely on it
- If you aren’t sure what action to take automatically, **tell a human**
- Stateful distributed systems and blind restarts do not mix
- Anytime you change how data is flowing, test in a stage environment if possible first
- Not all problems have a singular solution, use metrics to tease out the root cause before acting
### Troubleshooting (JMX) - Utilization/Saturation

<table>
<thead>
<tr>
<th>Key</th>
<th>Resource utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Replica Manager</strong></td>
<td></td>
</tr>
<tr>
<td>UnderReplicatedPartitions</td>
<td></td>
</tr>
<tr>
<td>RequestHandlerAvgIdlePercent</td>
<td></td>
</tr>
<tr>
<td><strong>Request Handler Pool</strong></td>
<td></td>
</tr>
<tr>
<td>NetworkProcessorAvgIdlePercent</td>
<td></td>
</tr>
<tr>
<td>RequestQueueSize</td>
<td></td>
</tr>
<tr>
<td>RequestQueueTimeMs</td>
<td></td>
</tr>
<tr>
<td><strong>Network Processor Threads</strong></td>
<td></td>
</tr>
<tr>
<td>LocalTimeMs</td>
<td></td>
</tr>
<tr>
<td>RemoteTimeMs</td>
<td></td>
</tr>
<tr>
<td>ResponseQueueSize</td>
<td></td>
</tr>
<tr>
<td>RequestsPerSec</td>
<td></td>
</tr>
<tr>
<td>ResponseSendTimeMs,</td>
<td></td>
</tr>
</tbody>
</table>
In Summary...

- Get those JMX metrics monitoring systems in place!
- Understand what your metrics are telling you before taking action
- Only restart if you have a reason to believe it will fix the problem
- When adding clients or brokers, test in a staging environment