POSTED ON JUL 19, 2018 TO DATA INFRASTRUCTURE, NETWORKING & TRAFFIC

Location-Aware Distribution: Configuring servers at scale

By Ali Haider Zaveri

Facebook's infrastructure consists of many geo-distributed data centers that host millions of servers. These servers run a number of systems, ranging from front-end web servers to News Feed aggregators to our messaging and live video applications. In addition to regular code pushes, we also push thousands of configuration changes to our servers every day. Thus it is fairly common for our servers to execute trillions of configuration checks. One example of a configuration check is a web server personalizing the user experience by displaying text in English for one user and in Portuguese for another.

In this post, we describe Location-Aware Distribution (LAD), a new peer-to-peer system that handles the distribution of configuration changes to millions of servers. We find that LAD is dramatically better at distributing large updates, 100 MB for LAD versus 5 MB previously, and also scales to support around 40,000 subscribers per distributor versus 2,500 subscribers before.

Background: how did we distribute configuration before LAD?

As described in our SOSP 2015 paper, Facebook's configuration management system (called Configerator) used ZooKeeper, an open source distributed synchronization service, to distribute configuration updates.

All data in ZooKeeper is stored in a single, consistent data store. Each ZooKeeper node (znode) may contain data or other child znodes. Each znode can be accessed using a hierarchical path (e.g., /root/znode/my-node). Clients reading data from ZooKeeper can set watches on znodes; when a znode is updated, ZooKeeper will notify those clients so that they can download the update.

ZooKeeper's strong consistency and strict ordering guarantees were critical in allowing us to scale and run our systems reliably. However, as our infrastructure grew to millions of machines, we found that ZooKeeper became a bottleneck.

- **Strict ordering guarantees:** A key feature of ZooKeeper is that it provides strict ordering guarantees, which means that writes are always processed in order by a single thread. Now, to ensure that reads aren't starved, ZooKeeper interleaves read and write processing. We found that updates to particular znodes could trigger a large number of watches, which in turn could trigger a large number of reads from the clients. A few such writes could result in stalls that block updates.
- **Thundering herds:** When a data znode is updated, ZooKeeper notifies all interested clients. This can trigger a thundering herd problem as incoming client requests overwhelm ZooKeeper.
- **Large updates can saturate NICs:** Some of our configuration files add up to 5 MB in size. Given a 10 Gb/s NIC, we found that a single box was limited to serving around 2,000 clients. If such a file is updated frequently, we would find that the update might take 10s of seconds to propagate to all interested clients.

Designing a system for the future

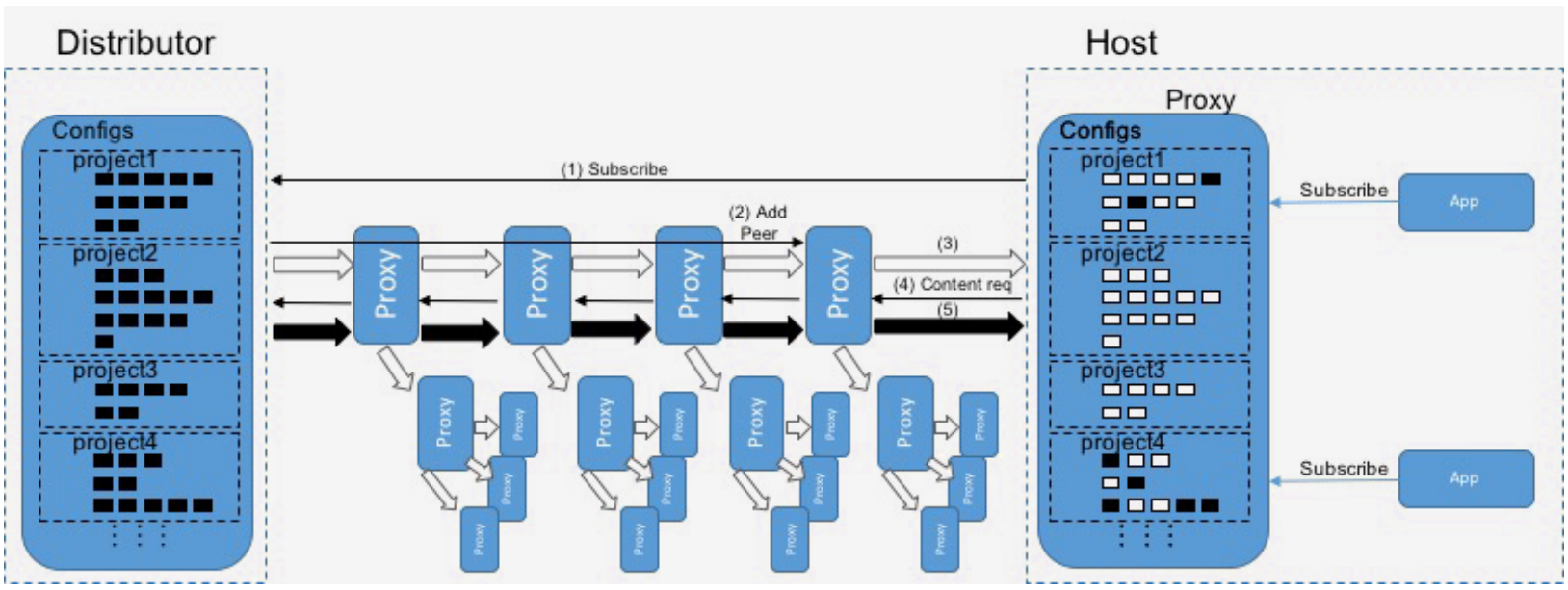
When we started designing a new distribution system, we came up with several requirements, including:

- Support for large updates: Increase the file size from 5 MB to 100 MB.
- Independent data store: ZooKeeper couples a data store with its distribution framework. We wished to separate the data storage and distribution components so we could size and scale each independently.
- Distribution capacity: Easily support millions of clients.
- Latency: Limit distribution latency to less than 5 seconds.
- Configuration files: Support 10x the number of configuration files as our prior ZooKeeper-based system.
- Resiliency against thundering herds and spikes in update rates.

Introducing Location-Aware Distribution (LAD)

LAD consists of two main components that communicate with each other via Thrift RPC:

- **Proxy:** A daemon running on every machine that serves configuration files to any interested application.
- **Distributor:** This component fulfills two roles. First, it polls the data store for new updates. Second, it constructs a distribution tree for a set of proxies interested in an update.



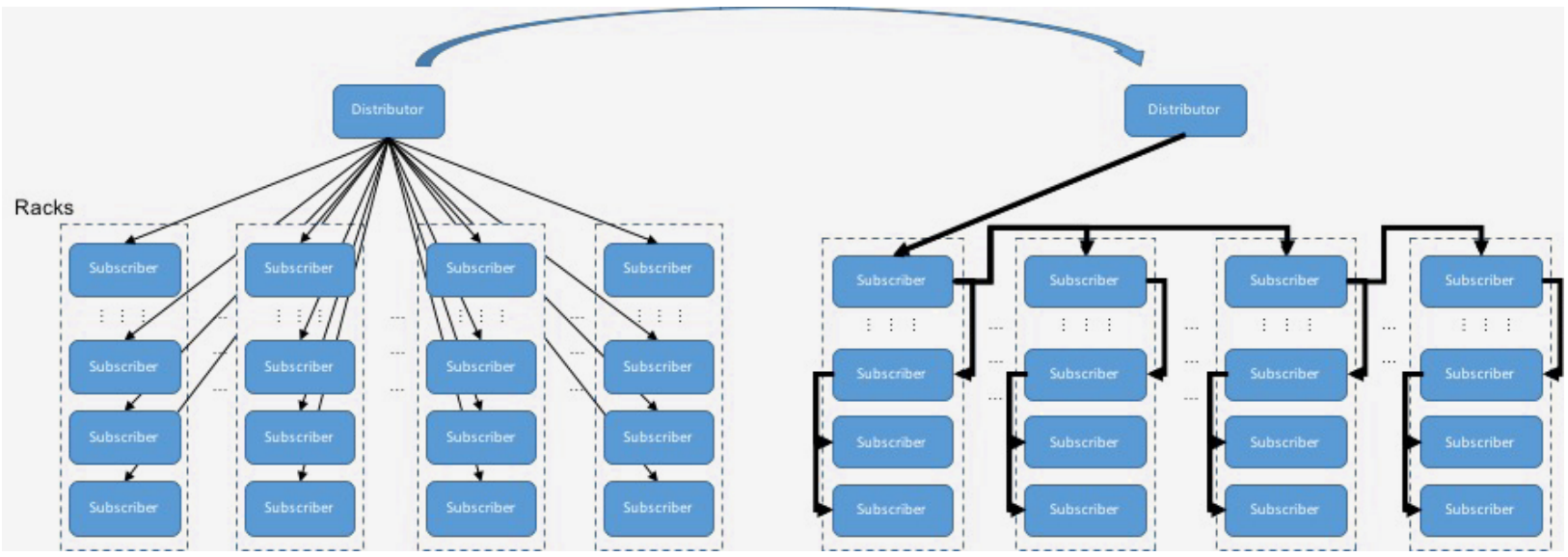
The above figure displays how LAD organizes proxies into a distribution tree, which is essentially a well-organized peer-to-peer network. As shown in step 1, the proxy sends "subscribe" requests to the distributor on behalf of the applications running on the box. The distributor adds the proxy to the distribution tree by sending an "add peer" request (step 2) to its parent. Once the proxy has been added to the tree, it starts receiving metadata (step 3) updates. These are filtered, and the proxy responds with content requests (step 4). If the parent has the content, it will send it to its child immediately; otherwise, it will download the content from its parent (step 5).

By leveraging a tree, LAD ensures that updates are pushed only to interested proxies rather than to all machines in the fleet. In addition, a parent machine can directly send updates to its children, which ensures that no single machine near the root is overwhelmed.

A key lesson from our prior experience with ZooKeeper was to separate metadata updates and distribution from content distribution. LAD's architecture relies on proxies' constantly receiving metadata updates. If every proxy were to receive all metadata updates, the volume of requests would be too high. We resolve this in LAD by leveraging shards: Rather than relying on serving the entire data tree as part of one distribution tree, we split it into smaller distribution trees, each of which serves some part of the data tree.

We implement the sharding design by having each shard be rooted at a particular point in the directory structure and including all the objects recursively beneath it. Sharding allows for a happy medium for subscriptions: It limits the total number of trees while balancing the amount of metadata each proxy receives.

Control and data flow



The control plane (at left in the image above) is a lightweight Thrift RPC between a distributor and each subscriber. It is used by the distributor to send tree commands that notify a parent about its new children and to check subscriber liveness. The control plane is also used by the subscriber to send subscription requests to the distributor. The distributor maps these requests to the shard this subscription belongs to and adds the subscriber into the distribution tree.

The data flow plane (at right in the image above) is the Thrift RPC, located between peers in the distribution tree, which does the heavy lifting. It is used by the parent to send metadata updates originating at the distributor to its children. The data flow plane is also used by the children to ask their parents for the content they are interested in.

Having a separate control and data flow plane enables each distributor to handle around 40,000 subscribers; ZooKeeper handled around 2,500 subscribers.

Lessons learned

We have learned several useful lessons in building and deploying LAD. Some highlights:

1. Tooling and monitoring is critical to production deployment. We learned that P2P based systems are challenging to operate and debug, as it is unclear what nodes are in the sending or receiving path for any given request.
2. Failure injection and disaster readiness testing is critical at scale. In addition to well-documented operational procedures, we have found it useful to run tests and develop strategies to react efficiently when issues arise. Specifically, we ran a battery of tests that introduced various types of application, host, network, and cluster-level failures to verify LAD's resiliency without affecting any clients. This was a fulfilling experience, as we found bugs as well as gaps in procedures and tooling.

3. Continuous and regular tests are critical to long-term system reliability. It is not enough to run the tests listed above only once, as things move fast at Facebook and assumptions about systems or tooling may not continue to hold true. We are automating our test processes so that we exercise our response to failures on a regular basis.

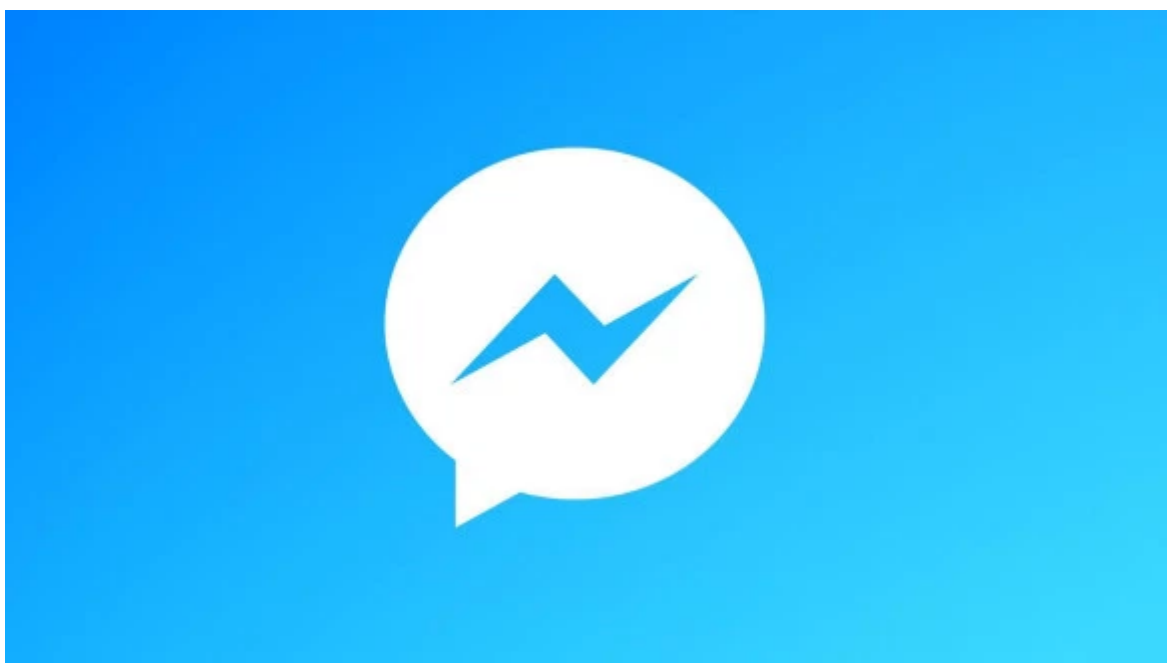
What's next for LAD?

LAD is currently being deployed into production as the data distribution framework for our configuration management system. We are also evaluating other applications for large-scale content distribution.

Related Posts

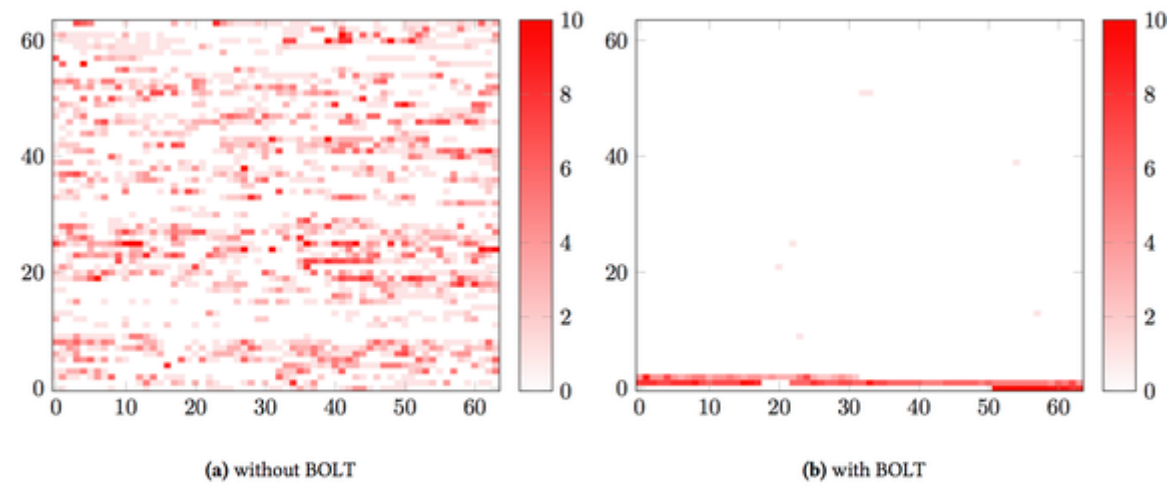
Jun 28, 2018

Spiral: Self-tuning services via real-time machine learning



Jun 26, 2018

Migrating Messenger storage to optimize performance



Jun 19, 2018

Accelerate large-scale applications with BOLT

Related Positions

Data Engineering Manager - Infra Strategy

MENLO PARK, US

Data Scientist, Infrastructure

MENLO PARK, US

Engineering Manager - Metrics & Infrastructure

ZURICH, SWITZERLAND

Data Engineer, Infrastructure Strategy

MENLO PARK, US

Manager, Infrastructure Data Science

BOSTON, US

See All Jobs

Join Our Engineering Community

Available Positions

Data Engineering Manager - Infra Strategy
MENLO PARK, US
Data Scientist, Infrastructure
MENLO PARK, US
Engineering Manager - Metrics & Infrastructure
ZURICH, SWITZERLAND
Data Engineer, Infrastructure Strategy
MENLO PARK, US
Manager, Infrastructure Data Science
BOSTON, US

See All Jobs

Stay Connected

Facebook Engineering

Like

@fbOpenSource

Follow

facebook research

Facebook Research

Like

Open Source

Facebook believes in building community through open source technology. Explore our latest projects in Artificial Intelligence, Data Infrastructure, Development Tools, Front End, Languages, Platforms, Security, Virtual Reality, and more.

ANDROID

iOS

WEB

BACKEND

HARDWARE

Learn More

Facebook Developers

Like

RSS

Subscribe