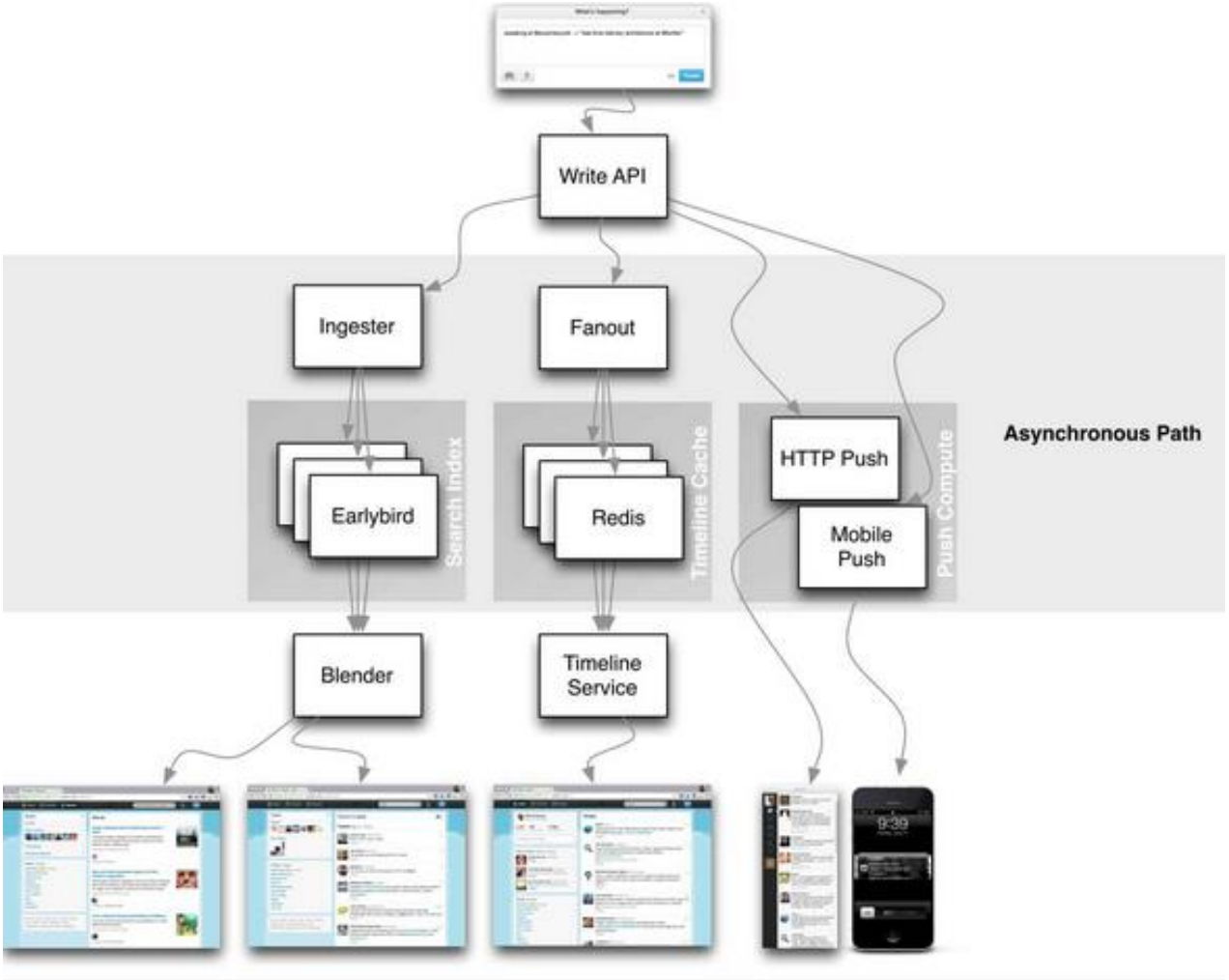
The Architecture of Twitter

Febrian Imanda Effendy
School of Computing, Telkom University
febrian.imanda@gamil.com



Groups

- Layanan platform *group* berperan sebagai inti skalabilitas infrastruktur dari twtiter.
    - Twitter menjalankan layanan timeline, tweet, *user*, *social graph*, dan semua kemampuan yang dimiliki oleh twitter sebagai platform.
    - *Internal clients* kurang lebih menggunakan API yang sama dengan *external clients.*
- Twitter memiliki sekelompok arsitektur yang mengurus arsitektur keseluruhan.

Push and Pull

- Pengguna membuat sebuah konten sepanjang waktu. Tugas dari twitter mencari tahu bagaimana menggabungkan konten terpublis. Bagaimana mengirimkan ini ke *follower.*
- Pengiriman ke *timeline cluster*, *push notifications, email,* hingga semua notifikasi iOS sama baiknya dengan Android, BBM dan SMS.
- Ada dua jenis *timeline* : yaitu *user timeline* dan *home timeline*
    - *User Timeline* adalah semua tweet yang ditulis secara personal
    - *Home Timeline* adalah semua tweet dari *user* yang kita *follow*
- Dasar *Pull*
    - *Timeline* sebagai target. Hal seperti twitter.com dan *home_timeline* API. Kita meminta data dari twitter melalui REST API.
- Dasar *Push*
    - Sistem twitter secara *real-time* memberikan tweet secepat 22MB/detik melalui Firehose.

Dasar Push Tingkat Tinggi Twitter Timeline

- Tweet diberikan melalui API. Ini melaju melalui penyeimbang beban dan *Twitter Front End* (TFE).
- Setelah proses fanout terjadi. Tweet yang datang ditempatkan secara masif di klaster Redis. Setiap tweet direplikasi sebanyak 3 kali di 3 mesin yang berbeda.
- *Query* fanout memberikan layanan graf berdasarkan Flock. Dimana tugas flock mengurus *follower* dan *following list.*
    - Flock memberikan sosial graf untuk penerima dan mulai mengurutkan sepanjang timeline yang disimpan di klaster redis.
    - Klaster redis memliki beberapa RAM seukuran *TeraBytes.*
    - Jika kita dikluar batas dari redis, maka kita akan melalui proses rekonstruksi dimana MySQL akan menangani penyimpanan disk menggunakan Gizzard.
    - Dengan mereplikasi tweet sebanyak 3 kali ketika mesin mendapatkan gangguan twitter tidak harus membuat ulang tweet ke semua timeline.

- Gizmoduck adalah layanan *user* dan TweetyPie adalah layanan objek Tweet. Setiap layanan memiliki *cache* mereka sendiri. *Cache* untuk *user* adalah klaster memcache yang memiliki seluruh basis *user* dalam cache. TweetyPie memiliki tweet sekitar bulan lalu yang disimpan dalam klaster memcache nya.

Pencarian secara Tingkat Tinggi

- Seketika tweet datang, Ingester memberi tanda dan mencari tahu semua yang ingin diberikan indeks dan menyatukan di mesin Early Bird.

- Blender membuat *timeline* pencarian. Dia menyatukan dan menyebarkan dari seluruh datacenter.
- Informasi aktivitas akan dikomputasi berbasi tulisan, menjadi sebuah *timeline* aktivitas. Seperti memilih tweet favorit dan membalas tweet semuanya akan diberikan sebuah ID. Semua aktivitas itu akan dimasukkan ke dalam Blender. Dalam perjalanan akan di komputasi ulang, digabungkan, hingga diurutkan.

Sebuah Pencarian dan *Pull* berkebalikan

- Pencarian dan *Pull* terlihat similar tetapi mereka memiliki atribut yang saling berkebalikan satu sama lainnya.
- Pada *Home Timeline :*
    - Ketika menulis, semua klaster redis menjadi pendukung, klaster flock menyimpan *user timeline* di *disk,* tetapi biasanya *timeline* ditemukan di klaster *r*edis*.*
    - Ketika membaca, melalu API atau web ini antara 0 dan 1 untuk menemukan mesin redis yang tepat.
- Pada *timeline* pencarian:
    - Untuk menulis, ketika tweet datang dan sampai di Ingester hanya sampai di satu mesin Early Bird.
    - Untuk membaca, pencarian tidak pernah sampai ke *disk*. Keseluruhan indeks Lucene terdapat di RAM. Jadi untuk menyebarkan maupun mengumpulkan yang akan dibaca menjadi efisien karena tidak pernah sampai di *disk*.

Dalam pemrosesan tweet yang ditulis didukung oleh Ruby dengan MRI, *single threaded server,* kemampuan proses terkikis habis ketika terdapat kebutuhan yang luar biasa. Twitter ingin melepas koneksi client secepat yang mereka mampu. Jadi ketika tweet datang, Ruby akan menaruhnya di dalam queue kemudian memutuskan koneksi.

Untuk memonitoring, twitter menggunakan sistem yang rapi dari Google yaitu Zipkin. Dengan ini twitter dapat meminta dan melihat setiap layanan yang didapat dibarengi detail permintaan waktu, jadi mereka bisa mendapat terperinci kinerja dari setiap permintaan.

Sumber :

http://highscalability.com/blog/2013/7/8/the-architecture-twitter-uses-to-deal-with-150m-active-users.html