hello!
gentle constructive rant
debugging large scale systems using events
understanding
system behaviour
app → events → column store

{ k: v }

analytical queries

SELECT ...
GROUP BY

users

😊
app

you are here

→

events

{k: v}

→

column store

→

analytical queries

SELECT ...

GROUP BY

→

users

😊
Software is becoming increasingly complex
Figure 3.15: Storage hierarchy of a WSC.

3.5.2 QUANTIFYING LATENCY, BANDWIDTH, AND CAPACITY

Figure 3.16 attempts to quantify the latency, bandwidth, and capacity characteristics of a WSC. For illustration we assume a system with 5,000 servers, each with 256 GB of DRAM, one 4 TB SSD, and eight 10 TB disk drives. Each group of 40 servers is connected through a 40-Gbps link to a rack-level switch that has an additional 10-Gbps uplink bandwidth per machine for connecting the rack to the cluster-level switch (an oversubscription factor of four). Network latency numbers assume a TCP/IP transport, and networking bandwidth values assume that each server behind an oversubscribed set of uplinks is using its fair share of the available cluster-level bandwidth. For disks, we show typical commodity disk drive (SATA) latencies and transfer rates.

The Datacenter as a Computer, Barroso et al
logs vs metrics: a false dichotomy
10.2.3.4 - - [1/Jan/1970:18:32:20 +0000] "GET / HTTP/1.1" 200 5324 ":" "curl/7.54.0" ":"
we can derive metrics from log streams
$ cat access.log
    | grep ... | awk ...
    | sort | uniq -c
{  
  time = "1970-01-01T18:32:20"
  status = 200
  method = "GET"
  path = ...
  host = "i-123456af"
  client_ip = "10.2.3.4"
  user_agent = "curl/7.54.0"
  request_dur_ms = 325
  request_bytes = 2456
  response_bytes = 5324
}
structured logs
summary events
canonical log lines
arbitrarily wide data blobs
events
A metric is an aggregation of events.
why do we aggregate?
count
p50
p99
max
histogram
SELECT ... 
GROUP BY 

users

analytical queries

column store

{ k: v }

events

app

😊

you are here
prometheus and the problem with metrics
"it's slow"
p99(request_latency) > 1000ms
300 requests were slow
... which ones?!
group by
Most monitoring questions are **top-k**.
🏆 top traffic by IP address
🏆 top resource usage by customer
🏆 top latency by country
🏆 top error count by host
🏆 top request size by client
how many users are impacted?
SELECT user_id, COUNT(*)
FROM requests
WHERE request_latency >= 1000
GROUP BY user_id
metrics will not tell you this
cardinality
http_requests_total{status=200} 10
http_requests_total{status=201}
http_requests_total{status=301}
http_requests_total{status=304}
...
http_requests_total{status=503}
user_id 10k
ip address space = $2^{32}$
4 billion possible values
kubectl get pods 100
build_id 100
the curse of dimensionality
{  
  status = 200  
  method = "GET"  
  path = ...  
  host = "i-123456af"  
  zone = "eu-central-1a"  
  client_ip = "10.2.3.4"  
  user_agent = "curl/7.54.0"  
  client_country = "de"  
  user_id = 30032  
  partition_id = 31  
  build_id = "9045e1"  
  customer_plan = "platinum"  
  endpoint = "tweet_detail"  
}
<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>200</td>
</tr>
<tr>
<td>method</td>
<td>&quot;GET&quot;</td>
</tr>
<tr>
<td>path</td>
<td>...</td>
</tr>
<tr>
<td>host</td>
<td>&quot;i-123456af&quot;</td>
</tr>
<tr>
<td>zone</td>
<td>&quot;eu-central-1a&quot;</td>
</tr>
<tr>
<td>client_ip</td>
<td>&quot;10.2.3.4&quot;</td>
</tr>
<tr>
<td>user_agent</td>
<td>&quot;curl/7.5.3&quot;</td>
</tr>
<tr>
<td>client_country</td>
<td>&quot;de&quot;</td>
</tr>
<tr>
<td>user_id</td>
<td>30032</td>
</tr>
<tr>
<td>partition_id</td>
<td>31</td>
</tr>
<tr>
<td>build_id</td>
<td>&quot;9045e1&quot;</td>
</tr>
<tr>
<td>customer_plan</td>
<td>&quot;platinum&quot;</td>
</tr>
<tr>
<td>endpoint</td>
<td>&quot;tweet_detail&quot;</td>
</tr>
</tbody>
</table>
\[
\begin{align*}
10 \times 5 \times 300 \times 20 \times 5 &= 172'800'000'000 \\
1k \times 300 \times 20 \times 1k \times 32 &= \\
10 \times 3 \times 20 &= \\
\end{align*}
\]

\[
= 172'800'000'000'
\]
app → events → column store

{ k: v }

you are here

SELECT ...
GROUP BY

analytical queries

users

😊
recording events
void do_rpc() {
    ...
    record_event({
        x: x,
        y: y,
        status: status,
        user: user,
        version: version,
        ...
    })
}
{  
  time = "1970-01-01T18:32:20"  
  status = 200  
  method = "GET"  
  path = "..."  
  host = "i-123456af"  
  region = "eu-central-1"  
  zone = "eu-central-1a"  
  client_ip = "10.2.3.4"  
  user_agent = "curl/7.54.0"  
  client_country = "de"  
  kernel = "5.0.0-1018-aws"  
  user_id = 30032  
  tweet_id = 2297111098  
  partition_id = 31  
  build_id = "9045e1"  
  request_id = "f2a3bdc4"  
  customer_plan = "platinum"  
  feature_blub = true  
  cache = "miss"  
  endpoint = "tweet_detail"  
  request_dur_ms = 325  
  db_dur_ms = 5  
  db_pool_dur_ms = 3  
  db_query_count = 63  
  cache_dur_ms = 2  
  svc_a_dur_ms = 32  
  svc_b_dur_ms = 90  
  request_bytes = 2456  
  response_bytes = 5324  
}

{

time = "1970-01-01T18:32:20"
status = 200
method = "GET"
path = ...
host = "i-123456af"
region = "eu-central-1"
zone = "eu-central-1a"
client_ip = "10.2.3.4"
user_agent = "curl/7.54.0"
client_country = "de"
kernel = "5.0.0-1018-aws"
}
{ 
    "user_id" = 30032,
    "tweet_id" = 2297111098,
    "partition_id" = 31,
    "build_id" = "9045e1",
    "request_id" = "f2a3bdc4",
    "customer_plan" = "platinum",
    "feature_blub" = true,
    "cache" = "miss",
    "endpoint" = "tweet_detail"
}
{  
  request_dur_ms = 325  
  db_dur_ms = 5  
  db_pool_dur_ms = 3  
  db_query_count = 63  
  cache_dur_ms = 2  
  svc_a_dur_ms = 32  
  svc_b_dur_ms = 90  
  request_bytes = 2456  
  response_bytes = 5324  
}
traces vs events: a false dichotomy
we can derive events from traces
Canopy: An End-to-End Performance Tracing And Analysis System

Jonathan Kaldor†    Jonathan Mace*    Michal Bejda†    Edison Gao†    Wiktor Kuropatwa†
Joe O’Neill†    Kian Win Ong†    Bill Schaller†    Pingjia Shan†    Brendan Viscomi†
Vinod Venkataraman†    Kaushik Veeraraghavan†    Yee Jiun Song†

†Facebook    *Brown University
stick those events in kafka
app → events → column store

{ k: v } → analytical queries

SELECT ...
GROUP BY

you are here

users

😊
Columnar Storage changed my life
### 2. WORKLOADS AND SOFTWARE INFRASTRUCTURE

Resource replication that has already been provisioned for fault-tolerance, thereby achieving small additional overheads for existing systems. They predict that tolerant techniques will become more invaluable in the next decade as we build ever more formidable online web services.

#### 2.6.5 LATENCY NUMBERS THAT ENGINEERS SHOULD KNOW

Table 2.3: Latency numbers that every WSC engineer should know. (Updated version of table from [Dea09].)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache reference</td>
<td>1.5 ns</td>
</tr>
<tr>
<td>L2 cache reference</td>
<td>5 ns</td>
</tr>
<tr>
<td>Branch misprediction</td>
<td>6 ns</td>
</tr>
<tr>
<td>Uncontended mutex lock/unlock</td>
<td>20 ns</td>
</tr>
<tr>
<td>L3 cache reference</td>
<td>25 ns</td>
</tr>
<tr>
<td>Main memory reference</td>
<td>100 ns</td>
</tr>
<tr>
<td>Decompress 1 KB with Snappy [Sna]</td>
<td>500 ns</td>
</tr>
<tr>
<td>“Far memory”/Fast NVM reference</td>
<td>1,000 ns (1us)</td>
</tr>
<tr>
<td>Compress 1 KB with Snappy [Sna]</td>
<td>2,000 ns (2us)</td>
</tr>
<tr>
<td>Read 1 MB sequentially from memory</td>
<td>12,000 ns (12 us)</td>
</tr>
<tr>
<td>SSD Random Read</td>
<td>100,000 ns (100 us)</td>
</tr>
<tr>
<td>Read 1 MB bytes sequentially from SSD</td>
<td>500,000 ns (500 us)</td>
</tr>
<tr>
<td>Read 1 MB sequentially from 10Gbps network</td>
<td>1,000,000 ns (1 ms)</td>
</tr>
<tr>
<td>Read 1 MB sequentially from disk</td>
<td>10,000,000 ns (10 ms)</td>
</tr>
<tr>
<td>Disk seek</td>
<td>10,000,000 ns (10 ms)</td>
</tr>
<tr>
<td>Send packet California→Netherlands→California</td>
<td>150,000,000 ns (150 ms)</td>
</tr>
</tbody>
</table>

Recently, cloud computing has emerged as an important model for replacing traditional enterprise computing systems with one that is layered on top of WSCs. The proliferation of high speed inter- The Datacenter as a Computer, Barroso et al.
• 1TB Hitachi Deskstar 7K1000

• disk seek time = 14ms

• transfer rate = 69MB/s

• 62.5 billion rows (= 1TB / 16 bytes)

• 28 years (= 62.5 billion rows * 14 ms/row / 32 × 10^9 ms/year)
• 1TB Hitachi Deskstar 7K1000

• transfer rate = 69MB/s

• 4 hours (= 1,000,000MB / 69MB/s / 3600 s/hour)
- **SSD**
  
  - transfer rate = 1GB/s
  
  - 15 minutes (= 1.000GB / 1GB/s / 60 s/min)
10GB
Dremel: Interactive Analysis of Web-Scale Datasets, Google
10 GB / 8 bytes per data point

= 1.3 billion events
time-based partitioning
dynamic sampling
it's lossy, but that's fine
vectorized processing
sequential scans
×
columnar layout
×
time-based partitioning
×
compression / sampling
×
vectorized processing
×
sharding
putting it all together
The diagram illustrates the flow of data from an app to a column store, through events, and analytical queries that produce users.

- The app generates events.
- Events are transformed into a dictionary format: \{ k: v \}.
- This dictionary is stored in a column store.
- The column store processes analytical queries, which select and group by users.
we need more of this in the monitoring space!
SELECT  user_id, COUNT(*)
FROM    requests
WHERE   status >= 500
GROUP BY user_id
ORDER BY COUNT(*) DESC
LIMIT 10
top-k

cardinality

events
Dremel: Interactive Analysis of Web-Scale Datasets

Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer,
Shiva Shivakumar, Matt Tolton, Theo Vassilakis
Google, Inc.
{melnik,andrey,jlong,gromer,shiva,mtolton,teov}@google.com

Scuba: Diving into Data at Facebook

Lior Abraham*
Vinayak Borkar
Daniel Merl
Subbu Subramanian

John Allen
Bhuwan Chopra
Josh Metzler
Janet L. Wiener

Oleksandr Barykin
Ciprian Gerea
David Reiss
Okay Zed

Facebook, Inc. Menlo Park, CA
• Dremel: Interactive Analysis of Web-Scale Datasets from Google, 2010

• Scuba: Diving into Data at Facebook from Facebook, 2016

• Canopy: An End-to-End Performance Tracing And Analysis System from Facebook, 2017

• Look at Your Data by John Rauser, Velocity 2011

• Observability for Emerging Infra by Charity Majors, Strange Loop 2017

• Why We Built Our Own Distributed Column Store by Sam Stokes, Strange Loop 2017

• The Design and Implementation of Modern Column-Oriented Database Systems by Abadi et al, 2013

• Designing Data-Intensive Applications by Martin Kleppmann, 2017

• Monitoring in the time of Cloud Native by Cindy Sridharan, 2017

• Logs vs. metrics: a false dichotomy by Nick Stenning, 2019

• Using Canonical Log Lines for Online Visibility by Brandur Leach, 2016

• The Datacenter as a Computer: Designing Warehouse-Scale Machines by Barroso et al, 2018