CockroachDB at Devsisters

ChangWon Lee
DevOps Engineer

Pierre Ricadat
Server Tech Lead

September 20, 2022

DEVSISTERS

Who are we?

- **Devsisters** – Korean game company founded in 2007
  - Launched CookieRun: Kingdom in January 2021
- **ChangWon Lee** 🇰🇷 – DevOps Engineer
  - In charge of provisioning, deploying and monitoring our CockroachDB cluster
- **Pierre Ricadat** 🇫🇷 – Server Tech Lead
  - Using CockroachDB for storing various pieces of game data

CookieRun: Kingdom

Multiplayer mobile RPG game

- Over 40 millions downloads since last year
- 350,000+ concurrent players
- > 50,000+ requests/sec

- Huge load from Day 1

Architecture

The Problem

- Large, ever-growing user state
- Every game action triggers a state change

- Updating the state at each action would lead to terrible performance
- Splitting the state into multiple tables would lead to terrible complexity

Architecture

Event Sourcing

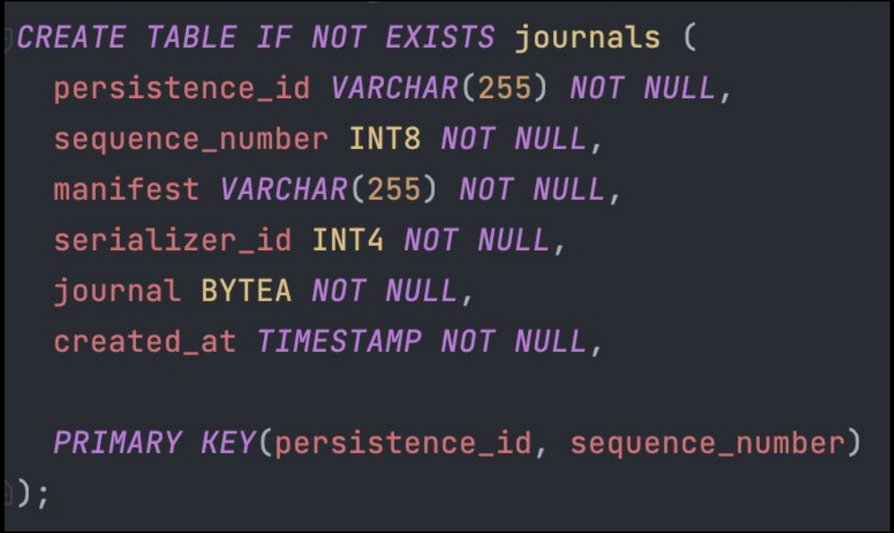
- Persist every event that happens to an entity
- Replay events to get the latest state of an entity

- Optimization: snapshots

Architecture

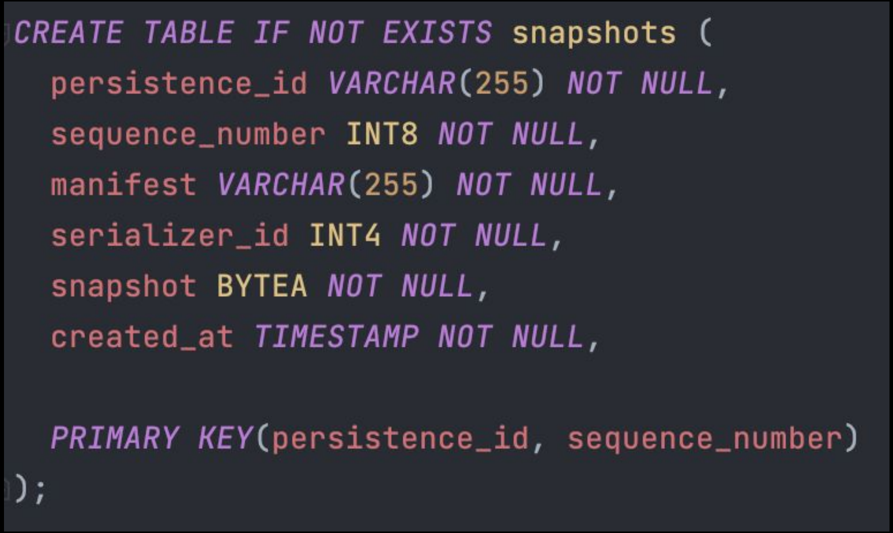
Event Sourcing

- Trivial DB Schema



```sql
CREATE TABLE IF NOT EXISTS journals (
  persistence_id VARCHAR(255) NOT NULL,
  sequence_number INT8 NOT NULL,
  manifest VARCHAR(255) NOT NULL,
  serializer_id INT4 NOT NULL,
  journal BYTEA NOT NULL,
  created_at TIMESTAMP NOT NULL,

  PRIMARY KEY(persistence_id, sequence_number)
);
```



```sql
CREATE TABLE IF NOT EXISTS snapshots (
  persistence_id VARCHAR(255) NOT NULL,
  sequence_number INT8 NOT NULL,
  manifest VARCHAR(255) NOT NULL,
  serializer_id INT4 NOT NULL,
  snapshot BYTEA NOT NULL,
  created_at TIMESTAMP NOT NULL,

  PRIMARY KEY(persistence_id, sequence_number)
);
```

Architecture

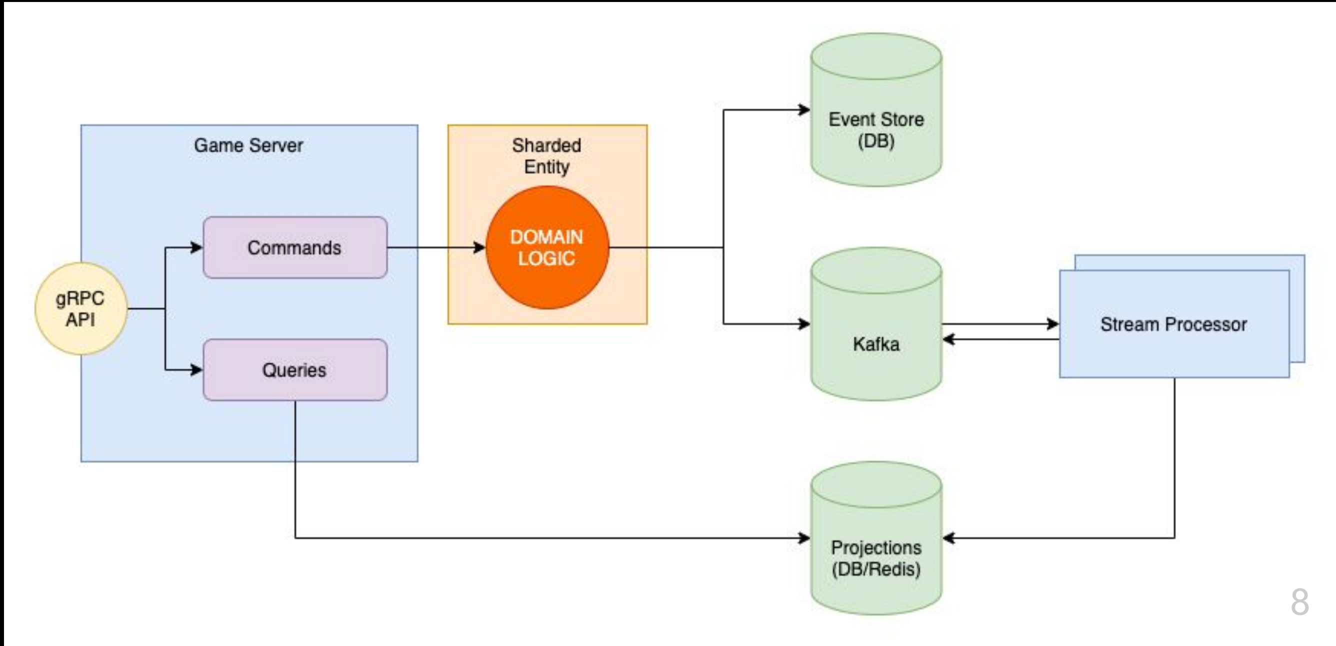
The Problem (2)

- Sometimes need to display a list of users with basic information
- Too slow to replay each user state

Architecture

CQRS (Command and Query Responsibility Segregation)

- Create “projections” for fast data access



DB Usage Patterns

Tables

- Small amount but very large ones
- Billions of rows
- Terabytes of data

- Reduced number of patterns

DB Usage Patterns

Queries

- Very INSERT-intensive
- Very little contention
- Occasional SELECT bursts

- The “midnight” problem

Launch Setup

Infrastructure

- 100% AWS, mostly on K8s
- CRDB on AWS EKS on EC2
- Custom Helm charts

- 24 m5d.8xlarge nodes, local NVMe storage
- 3 AZs without locality
- x7 replication factors

Disaster Outbreak 1

High Storage Pressure

- Launch on 01/21/21, survived the weekend
- Monotonous increase in user base since launch
- Did not fully consider side-effects of increasing RF
- -> High disk storage pressure

- Got back to work on Monday
- Expected <36h until disk full

Disaster Outbreak 1

High Storage Pressure

- “Let’s create a ballast file first, then scale out!”
- But misconfigured target directory as `/dev/<file_name>`
- Leading to Linux file system failure impacting 16 out of 24 nodes
- Lost another instance due to EC2 failure during recovery
- Only 7 out of 24 nodes remaining

Disaster Recovery 1

The “Unorthodox” Way

- Restoring to backup?

- Users already started growing affection to their cookies!

Disaster Recovery 1

The “Unorthodox” Way

- Reverse-engineered `.sst` files to analyze contained data
  - 7.6TB across remaining nodes
- `cockroach debug pebble sstable scan --filter $'\x04' .`
- Spark for sst2csv conversion, dedup, time sorting, etc.
- Two most important tables were mostly INSERT-only
  - If all MVCC data are recovered, we can recover “the most recent” user states

- 3.3% chance for replica to be gone for good

Disaster Recovery 1

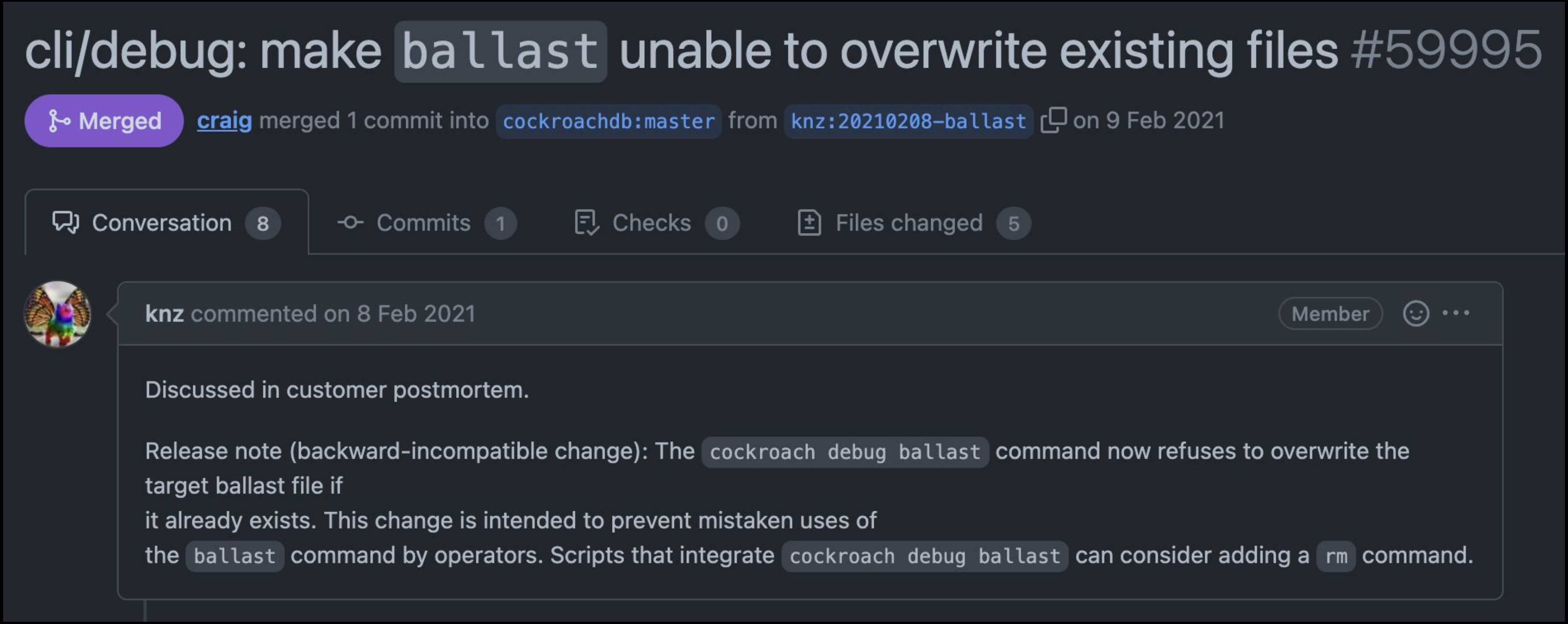
The “Unorthodox” Way

- CSV files from `cockroach debug pebble` and Spark
- Set up a secondary database cluster and import data
  - With increased capacity
- Data integrity check based on logs
- Rewrite server codes to reconstruct auxiliary tables

- Recovered 100% of data in 36h

Disaster Recovery 1

The Aftermath



cli/debug: make `ballast` unable to overwrite existing files #59995

Merged craig merged 1 commit into `cockroachdb:master` from `knz:20210208-ballast` on 9 Feb 2021

Conversation 8 Commits 1 Checks 0 Files changed 5

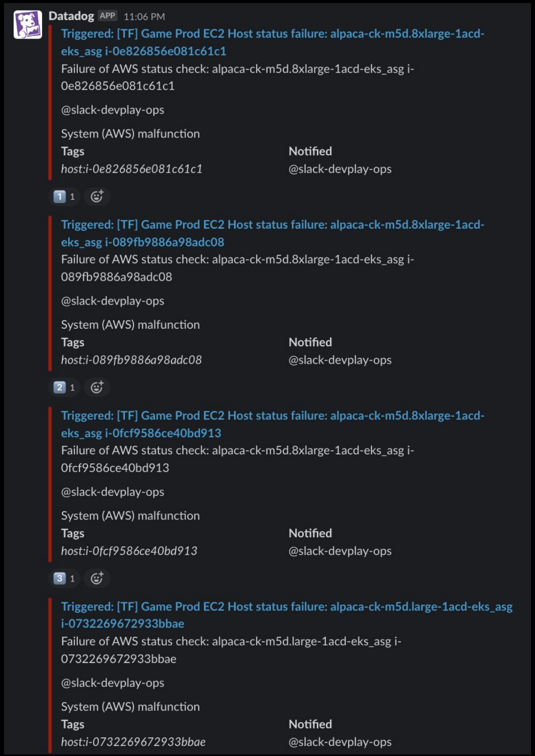
knz commented on 8 Feb 2021 Member

Discussed in customer postmortem.

Release note (backward-incompatible change): The `cockroach debug ballast` command now refuses to overwrite the target ballast file if
it already exists. This change is intended to prevent mistaken uses of
the `ballast` command by operators. Scripts that integrate `cockroach debug ballast` can consider adding a `rm` command.

Disaster Outbreak 2

The Black Swan Event



Datadog APP 11:06 PM

Triggered: [TF] Game Prod EC2 Host status failure: alpaca-ck-m5d.8xlarge-1acd-eks_asg i-0e826856e081c61c1
Failure of AWS status check: alpaca-ck-m5d.8xlarge-1acd-eks_asg i-0e826856e081c61c1

@slack-devplay-ops

System (AWS) malfunction

Tags	**Notified**
host:i-0e826856e081c61c1	@slack-devplay-ops

1️⃣ 1

Triggered: [TF] Game Prod EC2 Host status failure: alpaca-ck-m5d.8xlarge-1acd-eks_asg i-089fb9886a98adc08
Failure of AWS status check: alpaca-ck-m5d.8xlarge-1acd-eks_asg i-089fb9886a98adc08

@slack-devplay-ops

System (AWS) malfunction

Tags	**Notified**
host:i-089fb9886a98adc08	@slack-devplay-ops

2️⃣ 1

Triggered: [TF] Game Prod EC2 Host status failure: alpaca-ck-m5d.8xlarge-1acd-eks_asg i-0fcf9586ce40bd913
Failure of AWS status check: alpaca-ck-m5d.8xlarge-1acd-eks_asg i-0fcf9586ce40bd913

@slack-devplay-ops

System (AWS) malfunction

Tags	**Notified**
host:i-0fcf9586ce40bd913	@slack-devplay-ops

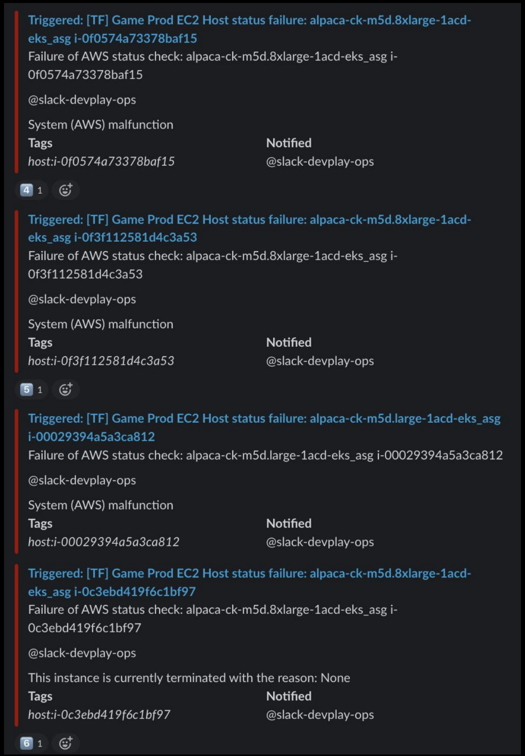
3️⃣ 1

Triggered: [TF] Game Prod EC2 Host status failure: alpaca-ck-m5d.large-1acd-eks_asg i-0732269672933bbae
Failure of AWS status check: alpaca-ck-m5d.large-1acd-eks_asg i-0732269672933bbae

@slack-devplay-ops

System (AWS) malfunction

Tags	**Notified**
host:i-0732269672933bbae	@slack-devplay-ops



Triggered: [TF] Game Prod EC2 Host status failure: alpaca-ck-m5d.8xlarge-1acd-eks_asg i-0f0574a73378baf15
Failure of AWS status check: alpaca-ck-m5d.8xlarge-1acd-eks_asg i-0f0574a73378baf15

@slack-devplay-ops

System (AWS) malfunction

Tags	**Notified**
host:i-0f0574a73378baf15	@slack-devplay-ops

4️⃣ 1

Triggered: [TF] Game Prod EC2 Host status failure: alpaca-ck-m5d.8xlarge-1acd-eks_asg i-0f3f112581d4c3a53
Failure of AWS status check: alpaca-ck-m5d.8xlarge-1acd-eks_asg i-0f3f112581d4c3a53

@slack-devplay-ops

System (AWS) malfunction

Tags	**Notified**
host:i-0f3f112581d4c3a53	@slack-devplay-ops

5️⃣ 1

Triggered: [TF] Game Prod EC2 Host status failure: alpaca-ck-m5d.large-1acd-eks_asg i-00029394a5a3ca812
Failure of AWS status check: alpaca-ck-m5d.large-1acd-eks_asg i-00029394a5a3ca812

@slack-devplay-ops

System (AWS) malfunction

Tags	**Notified**
host:i-00029394a5a3ca812	@slack-devplay-ops

Triggered: [TF] Game Prod EC2 Host status failure: alpaca-ck-m5d.8xlarge-1acd-eks_asg i-0c3ebd419f6c1bf97
Failure of AWS status check: alpaca-ck-m5d.8xlarge-1acd-eks_asg i-0c3ebd419f6c1bf97

@slack-devplay-ops

This instance is currently terminated with the reason: None

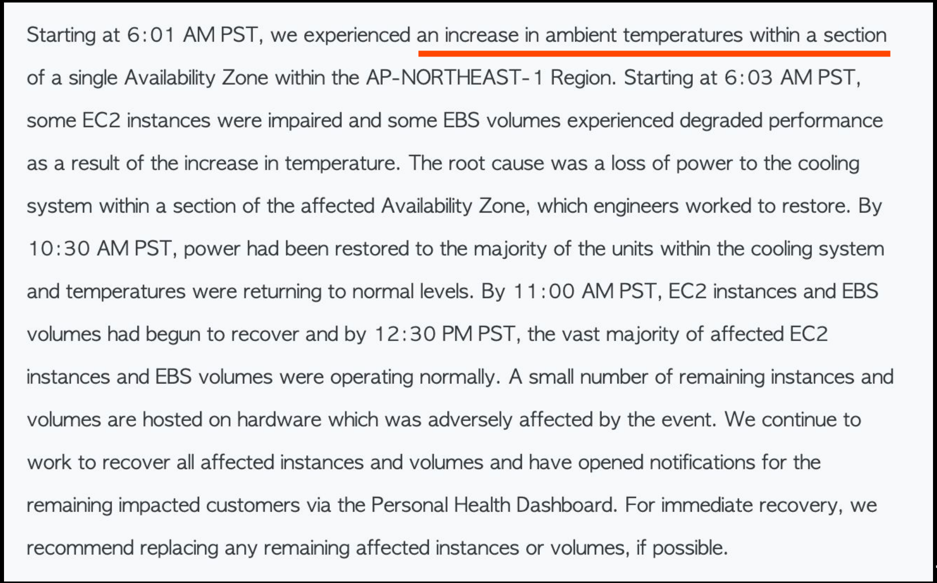
Tags	**Notified**
host:i-0c3ebd419f6c1bf97	@slack-devplay-ops

6️⃣ 1

Disaster Outbreak 2

The Black Swan Event

- AZ failure in AWS



Starting at 6:01 AM PST, we experienced an increase in ambient temperatures within a section of a single Availability Zone within the AP-NORTHEAST-1 Region. Starting at 6:03 AM PST, some EC2 instances were impaired and some EBS volumes experienced degraded performance as a result of the increase in temperature. The root cause was a loss of power to the cooling system within a section of the affected Availability Zone, which engineers worked to restore. By 10:30 AM PST, power had been restored to the majority of the units within the cooling system and temperatures were returning to normal levels. By 11:00 AM PST, EC2 instances and EBS volumes had begun to recover and by 12:30 PM PST, the vast majority of affected EC2 instances and EBS volumes were operating normally. A small number of remaining instances and volumes are hosted on hardware which was adversely affected by the event. We continue to work to recover all affected instances and volumes and have opened notifications for the remaining impacted customers via the Personal Health Dashboard. For immediate recovery, we recommend replacing any remaining affected instances or volumes, if possible.

Disaster Outbreak 2

The Price of Complacency

- Nodes lost: 6 / 60
- Range quorums lost: 34 / ~25000
    - Data integrity compromised

- Would not have happened had we configured locality

Disaster Recovery 2

The Price of Complacency

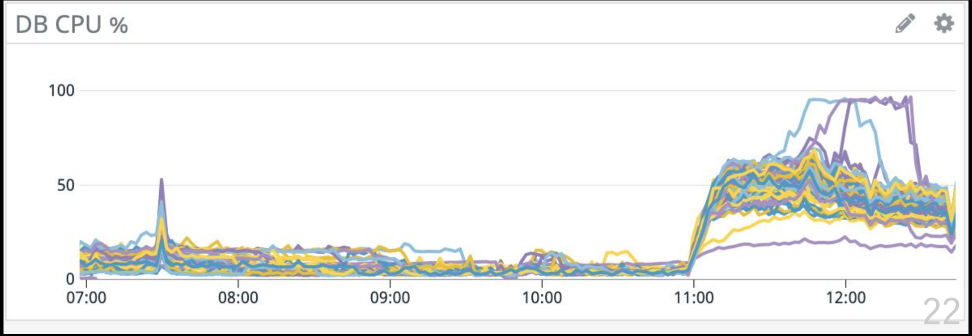
- Luckily, 32 ranges turned out to be CRDB system ranges
- Tried using `debug-unsafe-remove-dead-replica` to recover 2 ranges
    - Didn’t go well
- Did the same thing as DR1

- Recovered 100% data in 20h
- Now it seems there is `debug-recover-loss-of-quorum`

Disaster Outbreak 3

Hot Range Issue

- DB CPU surge in certain nodes after a game update
  - Occasional probe failures
- There was a hardware issue few days before
  - Thought similar issue was revisited
- Decommissioned problem nodes
- An hour later, all node became suspect nodes
- Taking backups exacerbated situation



Disaster Resolution 3

Hot Range Issue: Why?

- Changed primary key structure
- Users eagerly awaiting for new feature that used new table
  - Frequent row inserts
- Swapped a node for EC2 notice during maintenance
  - Replication queue already busy before service open

Things we learnt along the way

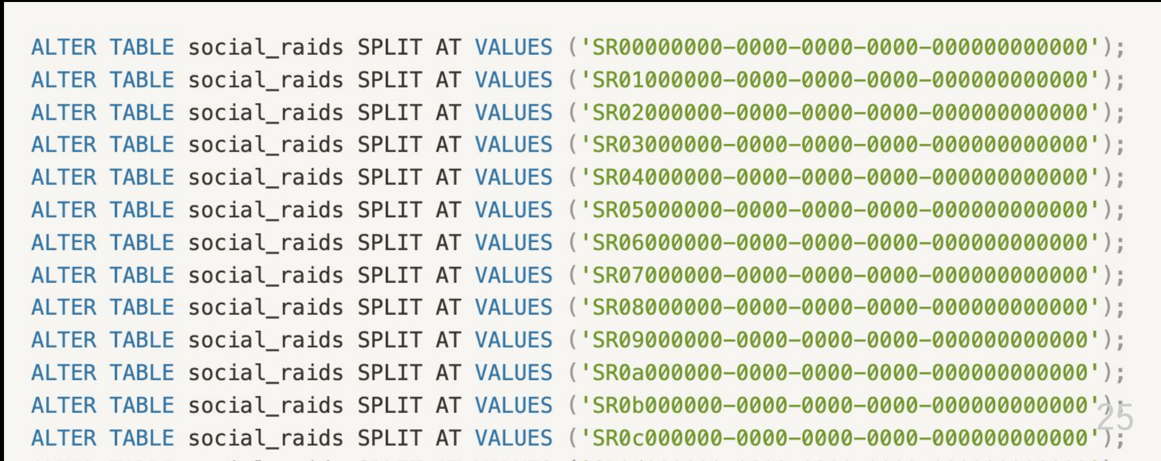
Infrastructure

- Capacity planning is important
- Ballast files
- Locality
- Hot ranges

Things we learnt along the way

Pre-splitting

- New table has only 1 range
- Immediate high traffic will cause a lot of range splitting
- `SPLIT AT / SCATTER` allows creating multiple ranges in advances



```sql
ALTER TABLE social_raids SPLIT AT VALUES ('SR00000000-0000-0000-0000-000000000000');
ALTER TABLE social_raids SPLIT AT VALUES ('SR01000000-0000-0000-0000-000000000000');
ALTER TABLE social_raids SPLIT AT VALUES ('SR02000000-0000-0000-0000-000000000000');
ALTER TABLE social_raids SPLIT AT VALUES ('SR03000000-0000-0000-0000-000000000000');
ALTER TABLE social_raids SPLIT AT VALUES ('SR04000000-0000-0000-0000-000000000000');
ALTER TABLE social_raids SPLIT AT VALUES ('SR05000000-0000-0000-0000-000000000000');
ALTER TABLE social_raids SPLIT AT VALUES ('SR06000000-0000-0000-0000-000000000000');
ALTER TABLE social_raids SPLIT AT VALUES ('SR07000000-0000-0000-0000-000000000000');
ALTER TABLE social_raids SPLIT AT VALUES ('SR08000000-0000-0000-0000-000000000000');
ALTER TABLE social_raids SPLIT AT VALUES ('SR09000000-0000-0000-0000-000000000000');
ALTER TABLE social_raids SPLIT AT VALUES ('SR0a000000-0000-0000-0000-000000000000');
ALTER TABLE social_raids SPLIT AT VALUES ('SR0b000000-0000-0000-0000-000000000000');
ALTER TABLE social_raids SPLIT AT VALUES ('SR0c000000-0000-0000-0000-000000000000');
```

Things we learnt along the way

Hot ranges

- Happens if you operate on data located on the same range

- Don’t index on sequential keys like timestamps!
- Be careful if you change the primary key pattern!

- Know how to find hot ranges (query, console)

Things we learnt along the way

Historical queries

- Sometimes you don’t need to display the “latest” data
- Contention can happen with long read queries

- Magic `AS OF SYSTEM TIME` clause

- You can even use it to recover lost data (we did a couple times!)
- Data available for amount of time defined by `gc.ttlseconds`

Thanks!