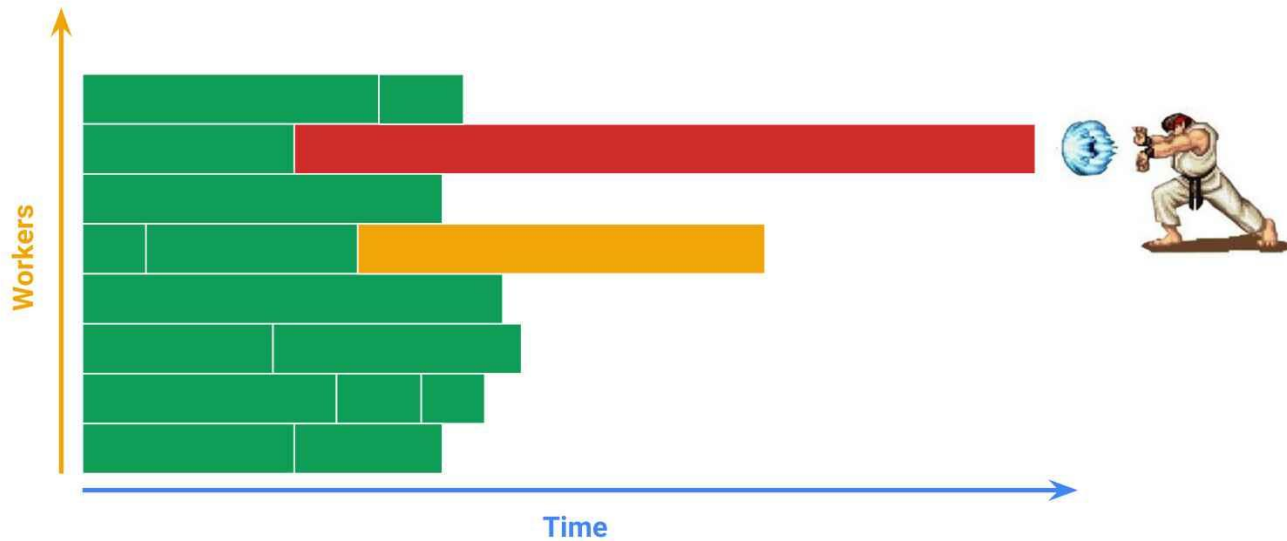


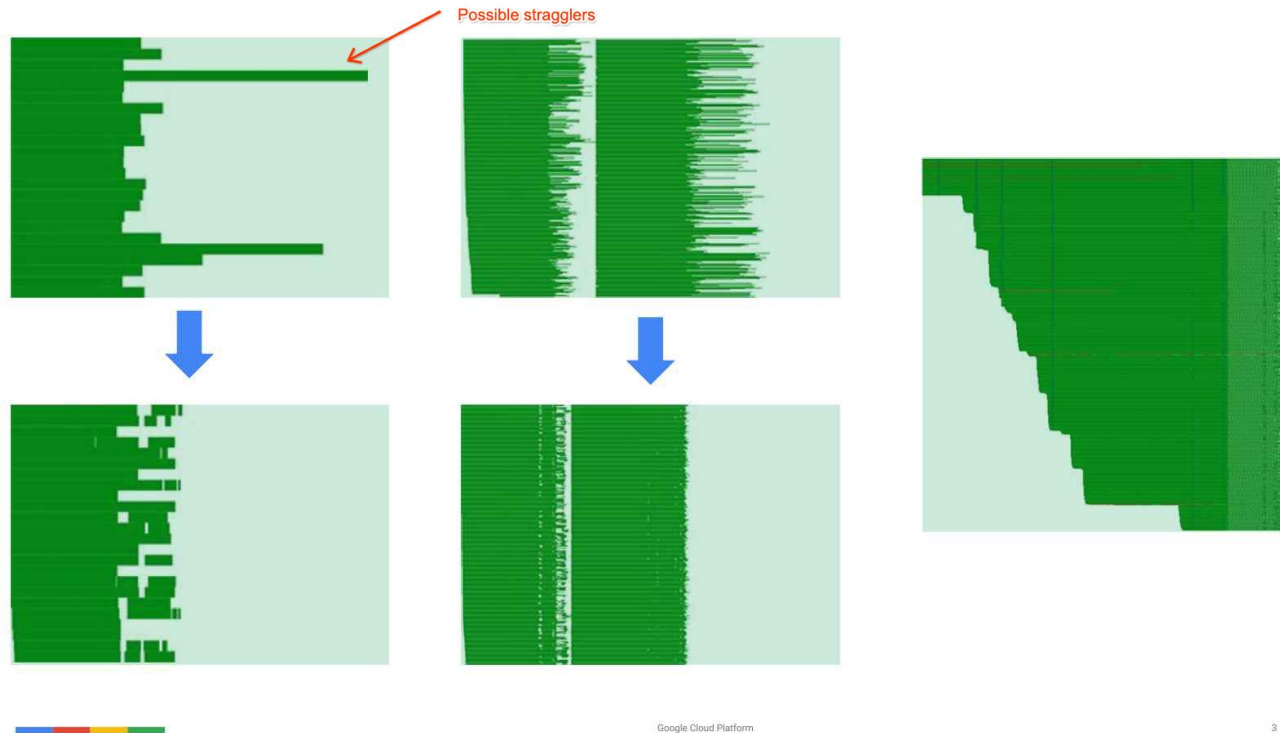
BEAM
SUMMIT

Hot keys detection in Apache Beam Pipelines

Shafiq Iqbal and Ikenna Okolo
Google



How stragglers can look like



WordCount

```
Pipeline p = Pipeline.create(options);
p.apply(TextIO.Read.from("gs://dataflow-samples/shakespeare/*"))
  .apply(FlatMapElements.via(
    word → Arrays.asList(word.split("[^a-zA-Z']+"))))
  .apply(Filter.byPredicate(word → !word.isEmpty()))
  .apply(Count.perElement())
  .apply(MapElements.via(
    count → count.getKey() + ": " + count.getValue()))
  .apply(TextIO.Write.to("gs://.../..."));
p.run();
```

Primitives to keep in mind

ParDo



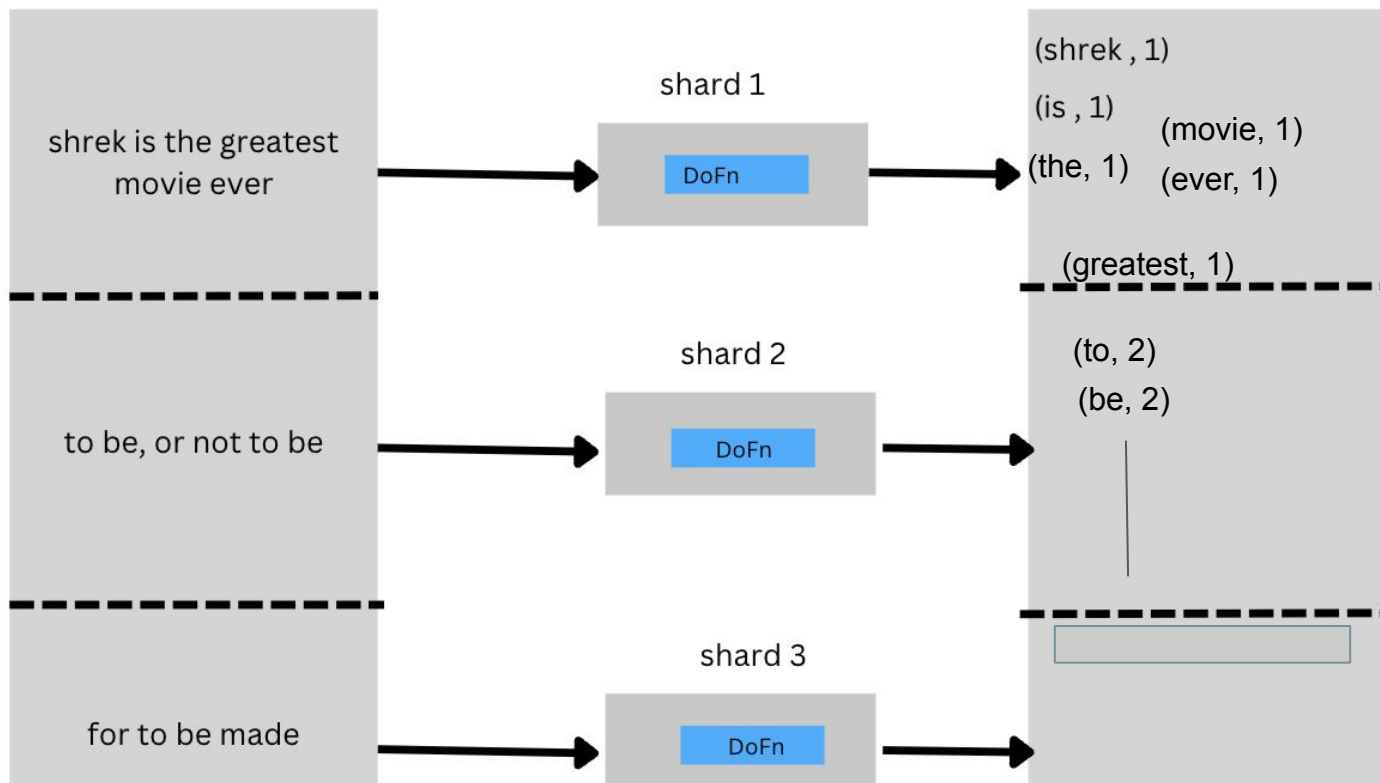
GroupByKey



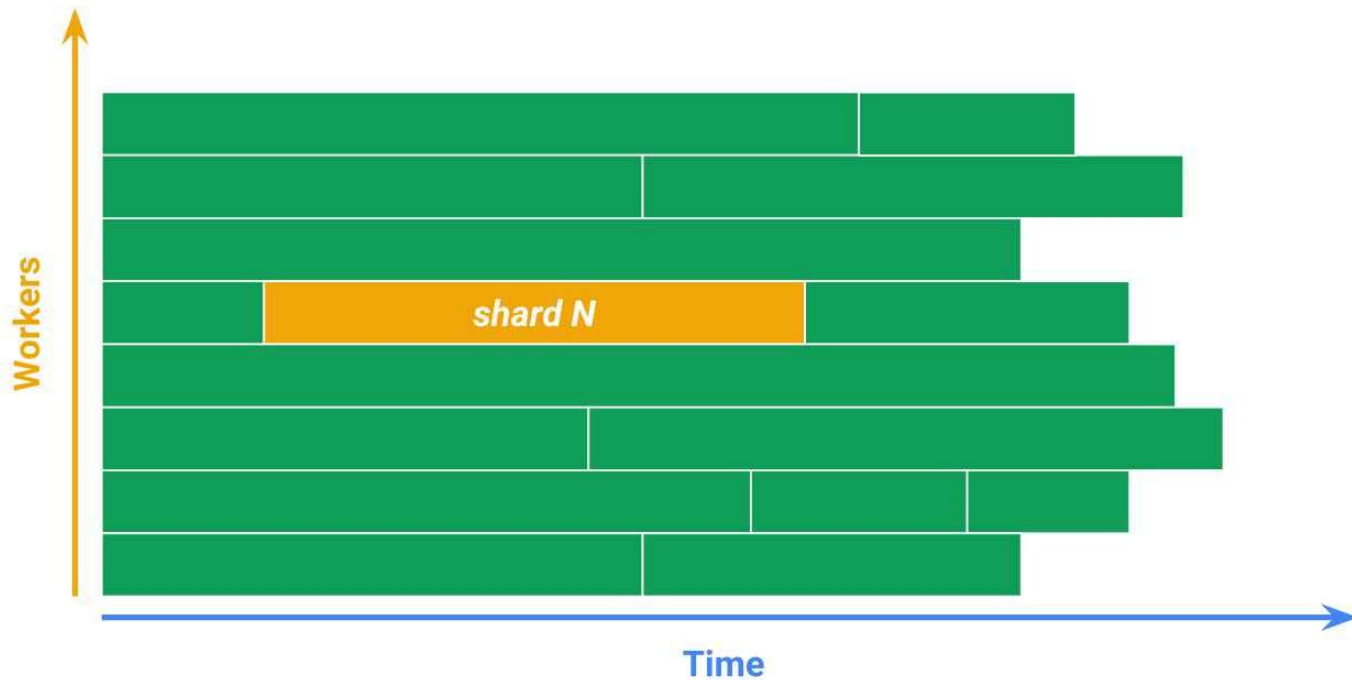
MapReduce = ParDo + GroupByKey + ParDo



How a ParDo would work

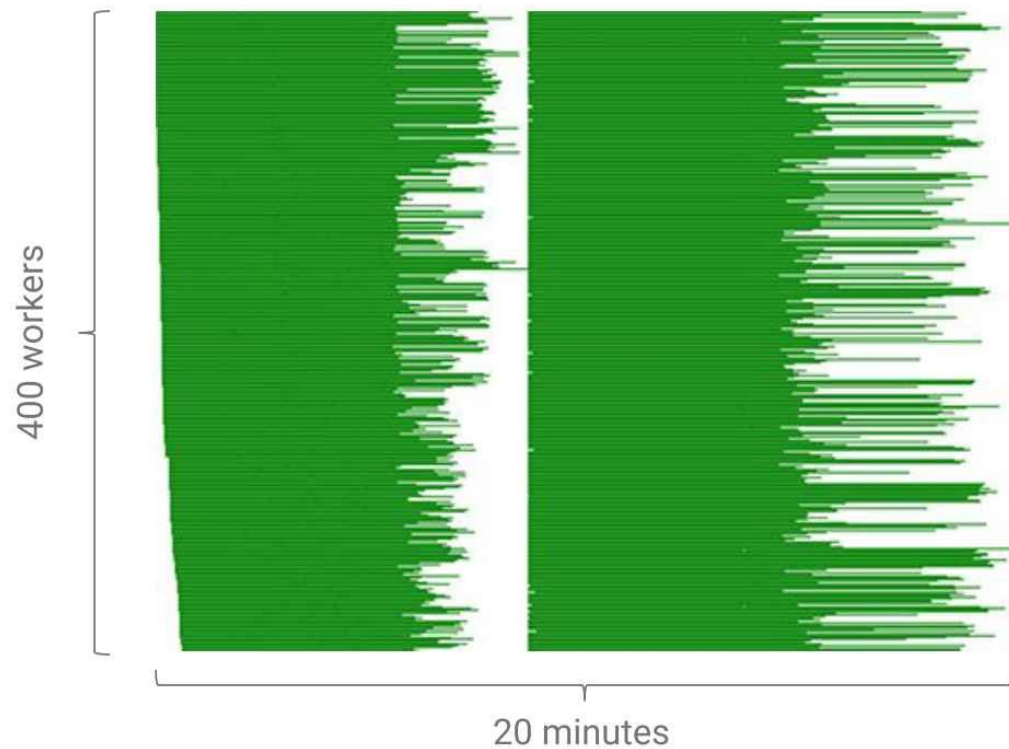


Gantt charts

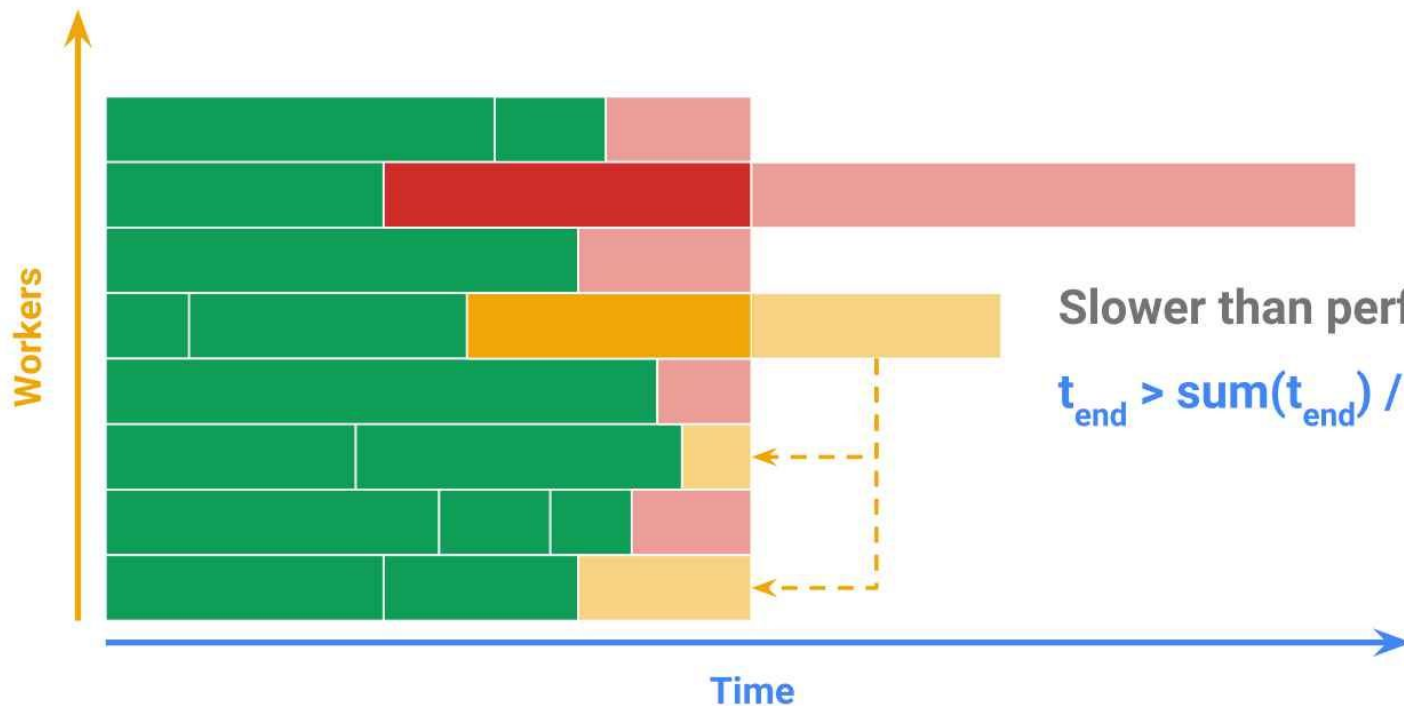


Large WordCount:

Read files, GroupByKey, Write files.



What is a straggler, really?



Amdahl's law: it gets worse at scale

$$\textit{Speedup} = \frac{N}{1 + (N-1) \cdot S}$$

#workers

serial fraction

Higher scale \Rightarrow More bottlenecked by serial parts.

Reasons for Stragglers

Uneven partitioning

- Process dictionary in parallel by first letter
-> 6x speedup only by ahmdahl's law

Uneven Complexity

- Join keys with some external input values

Uneven resources

- Bad machines, network or resource contention

Bugs

- Slow RPCs or bugs

Reasons for Stragglers

Hot keys



Uneven partitioning

- Process dictionary in parallel by first letter
-> 6x speedup only by ahmdahl's law

Uneven Complexity

- Join keys with some external input values

Uneven resources

- Bad machines, network or resource contention

Bugs

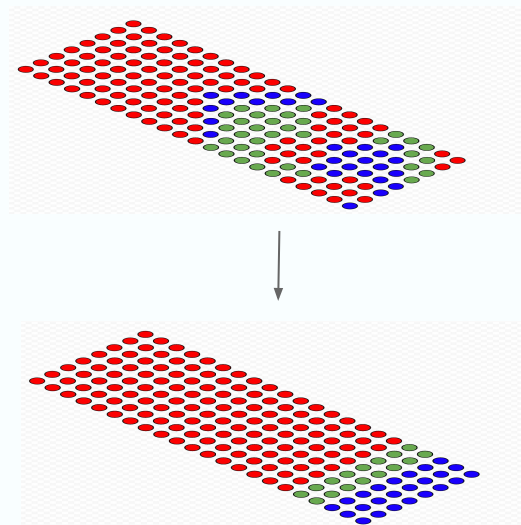
- Slow RPCs or bugs

What are hotkeys

A hot key is a key with enough elements to negatively impact pipeline performance. These keys limit a Pipeline's ability to process elements in parallel, which increases execution time.

Think about hotkeys in this way. Let's imagine there's a room filled with 150 Red, 30 Blue and 20 Green unsorted plates and there are 3 students who are to arrange those plates in sorted orders (as seen here to the right).

Let's assume that student 1 will sort the Red plates, student 2 will sort the blue and the last student will sort the green plates.



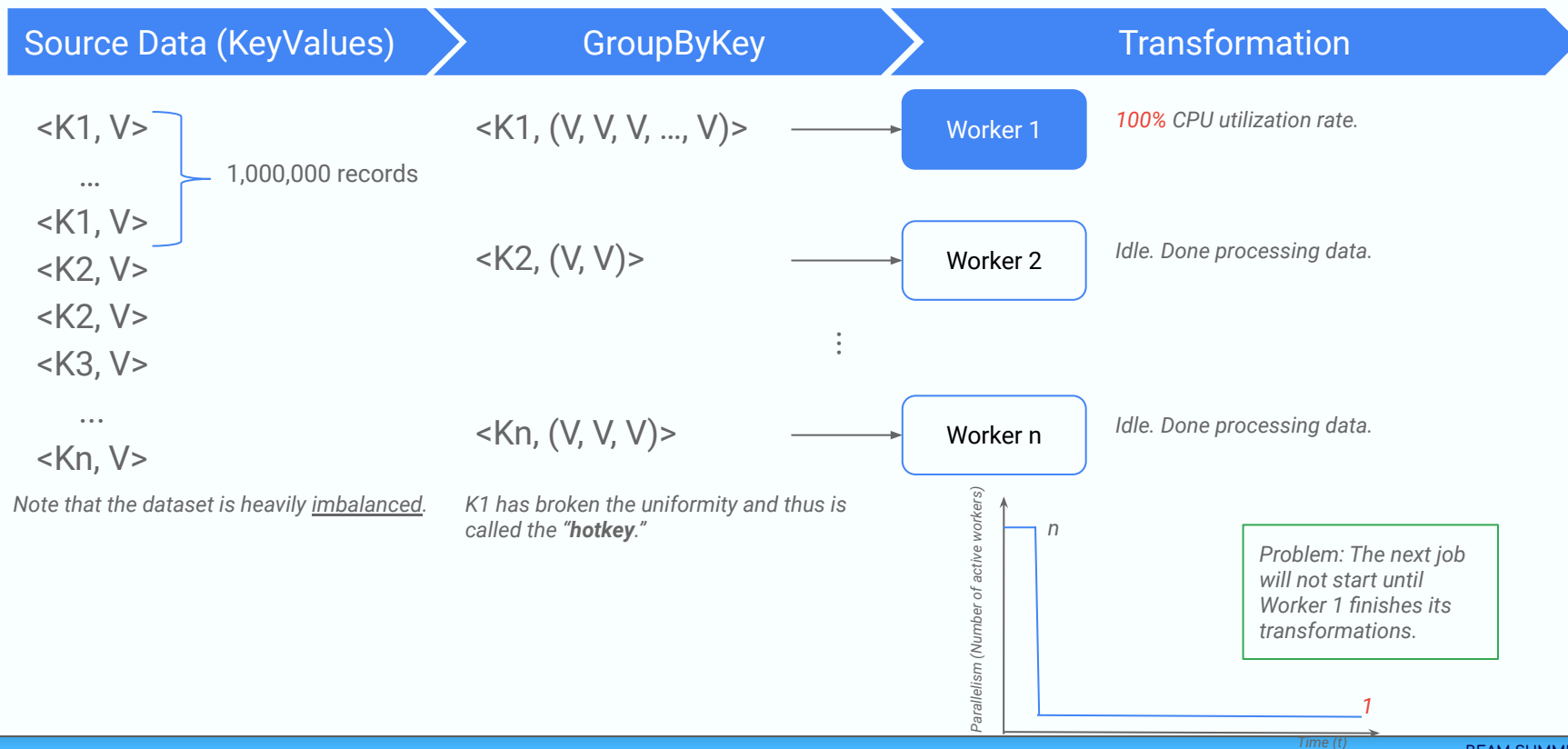
What are hotkeys contd...

From the illustration in the previous slide, students 2 and 3 will finish before student 1. Though the second and third students had already completed sorting their respective colored plates, they have to wait for the first student to complete theirs before the task can be termed as completed. This delay by student 1 is due to the larger number of plates they need to sort. In parallel processing, this is referred to as hotkeys.

If we replace the students with workers and the unsorted-plates with work-items to be processed, we can apply the same thinking to Dataflow pipelines. If the work-items are not evenly distributed, then there's bound to be an issue of hotkeys which obviously would impact the performance of the Pipeline.

In subsequent slides, we will explain this using a Key Value pair to represent individual work-items.

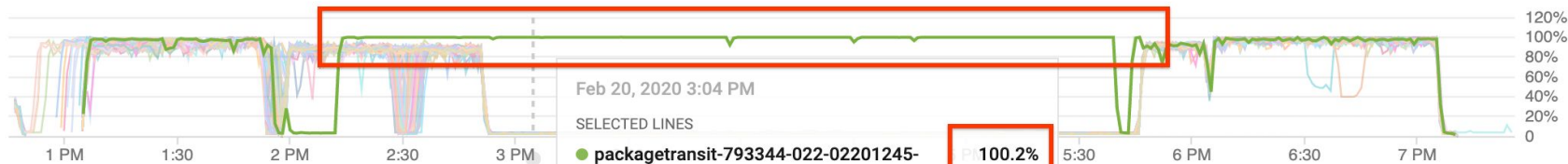
How do Hotkeys cause problems?



How to identify hotkeys

CPU utilization (All Workers) ▾ ?

Create alerting policy

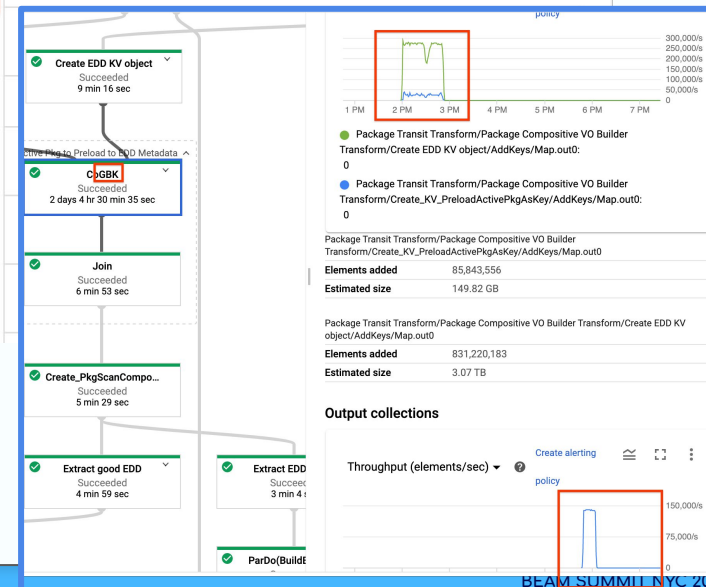


Feb 20, 2020 3:04 PM

SELECTED LINES

●	packagetransit-793344-022-02201245-bshe-harness-rbwz	100.2%
●	packagetransit-793344-022-02201245-bshe-harness-rbwz	100.2%
●	packagetransit-793344-022-02201245-bshe-harness-rgm8	4.19%
●	packagetransit-793344-022-02201245-bshe-harness-dggv	3.63%
●	packagetransit-793344-022-02201245-bshe-harness-rqgx	3.63%

100.2%



One of the quickest ways to identify a Job that is impacted by hotkeys is by taking a quick look at the worker CPU utilisation. While some workers are maxing out at about 90% utilisation, some are idle at about <5% utilisation. This truly indicates that there is a possibility that the Job is stuck due to hotkeys.

What can you do?

Uneven partitioning

Uneven complexity

Uneven resources

Noise

Oversplit

Hand-tune

Use data statistics

Backups

Restarts

Predictive

Weak

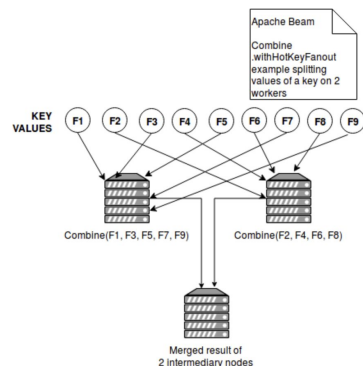
Data Monitoring, key partitioning,
iterative optimization

Using statistical analysis to pre-detect the hot key

How do customers fix hotkeys?

To resolve this issue, you may have to inform the customer to check that their data is evenly distributed. If a key has disproportionately many values, consider the following courses of action:

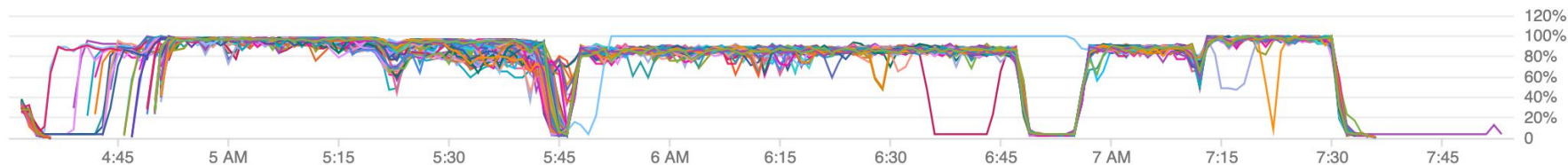
- Rekey their data. Apply a [ParDo](#) transform to output new key-value pairs.
- Autosharding
- `Combine.Globally #withFanout(int fanout)`
- Java jobs should consider using the [Combine.PerKey.withHotKeyFanout](#) transform.
- Python jobs should consider using the [CombinePerKey.with_hot_key_fanout](#) transform.
- Finally, consider enabling [Dataflow Shuffle](#) (if using dataflow).



Job not impacted by hotkeys anymore!

CPU utilization (All Workers) ▾ ?




[Create alerting policy](#)



Name

Value



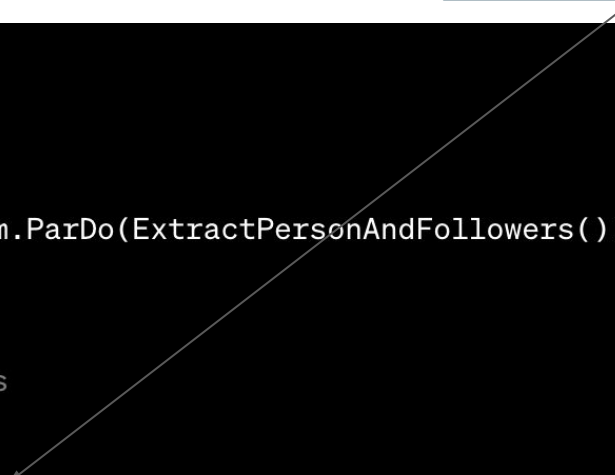
 dna-integration-gsi-edd-p-02270430-b79j-harness-0100	1.66%
 dna-integration-gsi-edd-p-02270430-b79j-harness-05z9	1.61%
 dna-integration-gsi-edd-p-02270430-b79j-harness-0g4x	2.08%
 dna-integration-gsi-edd-p-02270430-b79j-harness-0kjp	1.81%

A sample beam pipeline that sums total followers of every user

Potential Hot Key for users
with huge following

```
# Extract the person and their followers
person_followers = (
    following_list
    | "ExtractPersonAndFollowers" >> beam.ParDo(ExtractPersonAndFollowers())
)

# Group by person and sum their followers
aggregated_followers = (
    person_followers
    | "GroupByPerson" >> beam.GroupByKey()
    | "SumFollowers" >> beam.Map(lambda kv: (kv[0], sum(kv[1])))
)
```



How to solve it:

```
# Apply the fanout operation to handle hot keys
person_followers_with_hotkey_fanout = (
    person_followers
    | "WithHotKeyFanout" >> beam.transforms.util.WithHotKeyFanout(
        lambda person, followers: 1000 if followers >= 1000000 else 1
    )
)
```

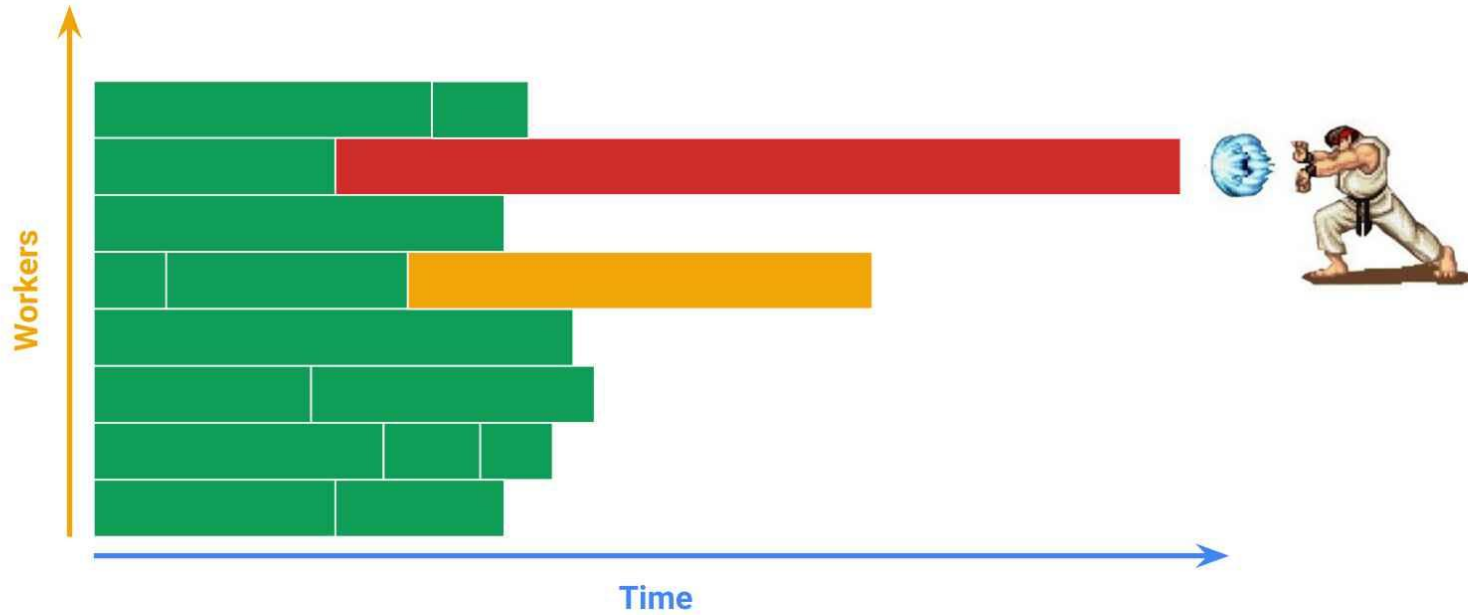
← If the hot key is extremely hot,
pre-aggregate data on multi-stage

```
# Use a combiner to sum followers by person
aggregated_followers = (
    person_followers
    | "CombineFollowers" >> beam.CombinePerKey(sum)
)

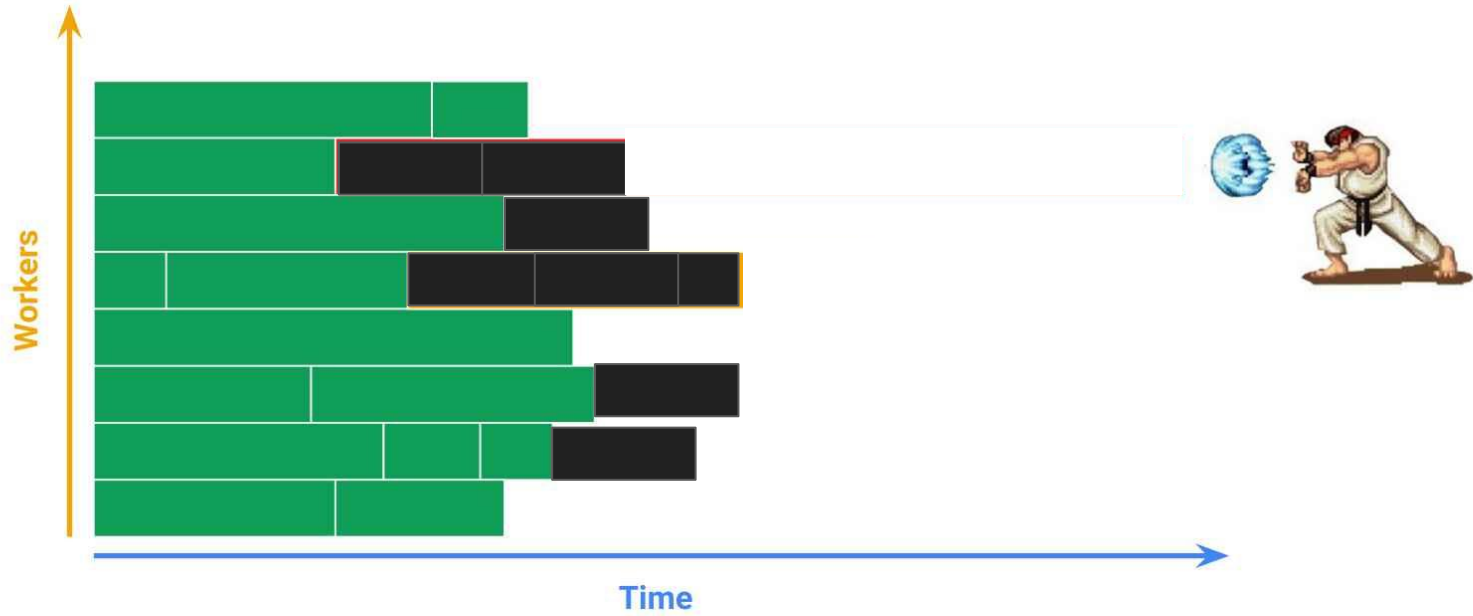
# Format and write the output
formatted_output = (
    aggregated_followers
    | "FormatOutput" >> beam.Map(lambda kv: f"Person: {kv[0]}, Total Followers: {kv[1]}")
    | "WriteOutput" >> beam.io.WriteToText("output.txt")
)
```

← Use a combiner to sum
followers by person

From this



To this



Hotkeys FAQ

Can we assign a more powerful machine to the worker that is processing the hotkey (i.e. Worker 1)?

>> Unfortunately, you cannot. Dataflow, by design, assigns the same machine to *all* of its workers.

*In that case, if all workers run with powerful machines, the pipeline will finish quicker.
+ It will be cheap, since most of them will be idle anyways.*

>> This will not speed up the process. A powerful machine will still use up only one of its cores. Imagine a giant for-loop to better understand -- cores do not split the work of a for loop.

I enabled autoscale, but my job doesn't finish any faster. Why?

>> You will see in monitoring that the average CPU utilization rate is far below 20%; therefore, Dataflow will not bring in more workers. Even if it does, it won't help -- remember that you already have n-1 idle workers. Surely n idle workers won't make a difference.

Root cause: dataset is imbalanced.

Fix the root cause: balance the dataset.

Solution: Classify the imbalanced key and break them down into smaller pieces.

<K1, V>	→	<K1.1, V>
...		<K1.1, V>
<K1, V>		<K1.2, V>
<K2, V>		<K1.2, V>
<K2, V>		...
<K3, V>		<K2, V>
...		...
<Kn, V>		<Kn, V>

Hotkeys FAQ

I can't see the hot key in logging

>> Turn the flag “hotkeyloggingenabled” and make sure the org hasn't put restrictions on the amount of logs available

I didn't define any keys in my pipeline, still facing hot keys

Root cause: dataset is imbalanced.

Fix the root cause: balance the dataset.

Solution: Classify the imbalanced key and break them down into smaller pieces.

<K1, V>	→	<K1.1, V>
...		<K1.1, V>
<K1, V>		<K1.2, V>
<K2, V>		<K1.2, V>
<K2, V>		...
<K3, V>		<K2, V>
...		...
<Kn, V>		<Kn, V>