Google

Managing Private Clouds with Flex

walfredo@google.com

2024-05-31

JSSPP 2024

Agenda

- The Private Cloud Management Problem
- Flex: Google's solution for this problem
  - Flex value proposition
- Resource Distribution
  - Pooling, Scaling
- Tiering and Product Design
- Governance and Obligation

The Cloud Provider problem

- Warehouse computing enables us to sell compute on demand
- Cheaper (economy of scale) and better (no worries about infrastructure) for the **user**

- Users have the expectation that the cloud will run their application
- Users hate toil
- Ideally, it should "just work"

- The Cloud Provider Problem is
  - How to best provision and manage the cloud to efficiently meet user expectation?
  - Which products to sell?

Running a cloud business

- Cloud products are built on physical resources
- Physical resources can be configured to sell different products
  - One may sell 1000 VMs
  - Or 500 VMs + 300 Dataflows
  - Or 400 VMs + 100 Dataflows + 200 App Engines

- The **planner** is responsible for the physical resources in the Cloud
  - Adding physical resources to the Cloud has lead time

- The **operator** is responsible for configuring the existing physical resources into Cloud products, as to best meet user expectations

Private vs public clouds

- In a Public Cloud, providers (planners and operators) compete with each other to entice users and maximize their profit
- In a Private Cloud, planners, operators and users are in the same company, which changes the problem

- Charge/Revenue Neutrality replaces Profit & Loss
- There is much greater control of the workload
  - Broader, more complex interfaces are okay, if more efficient
  - Emergency response can "harm" workload
- Fraud worries are greatly minimized
- Private clouds are typically a monopoly

A solution: Usage Based Chargeback

- Users show up and start using the system. They are charged for what they use.
- **Pros**:
  - The perfect user experience (when it works 🙂)
  - Shares buffer (which makes it more efficient)
- **Cons**:
  - As there is no signal of what the user will consume, it is hard to meet expectations efficiently
  - When we run out of shared buffer, we violate user expectations in an arbitrary manner (e.g. network congestion in a cluster)

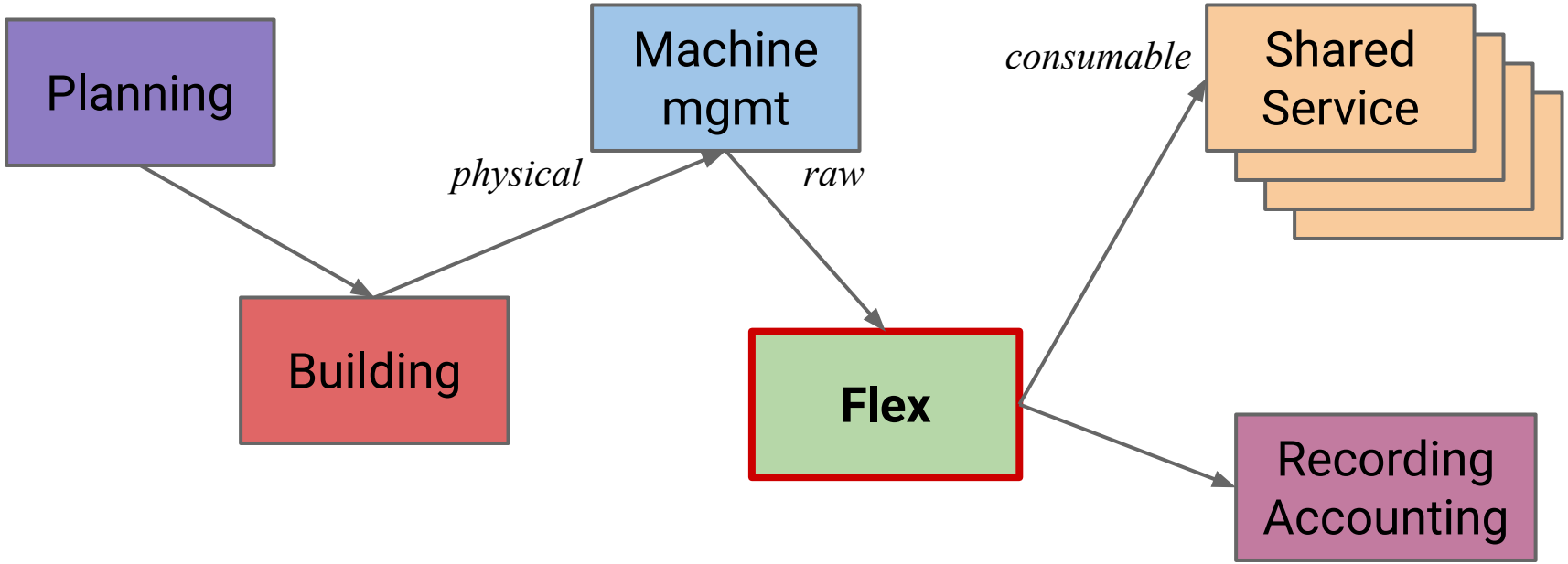
Another solution: Fixed Quota

- Capacity is sliced into quota, which can be transferred and subleased among users
- **Pros**:
  - The user expectation is very clear: You are going to be able to use your quota
- **Cons**:
  - High toil for users and operators alike
  - Inefficient as quota siloes resources that are estimated with (a big) buffer
  - Running out of quota can be catastrophic

Flex: Google's Solution

- Flex tries to strike a balance between Fixed Quota and Usage Based
- We approximate the "usage based" experience to most users
  - Most users are small and consume relatively little resources
  - A few large users need to engage in forecast and planning
- We seek efficiency by sharing buffers
- If we run out of resources, the behavior is pre-established
  - In **stock-out**, applications already running keep running, whereas new applications are prevented from starting
  - "It is much better to delay a launch then to bring down existing services" -- Balaji Venkatachari

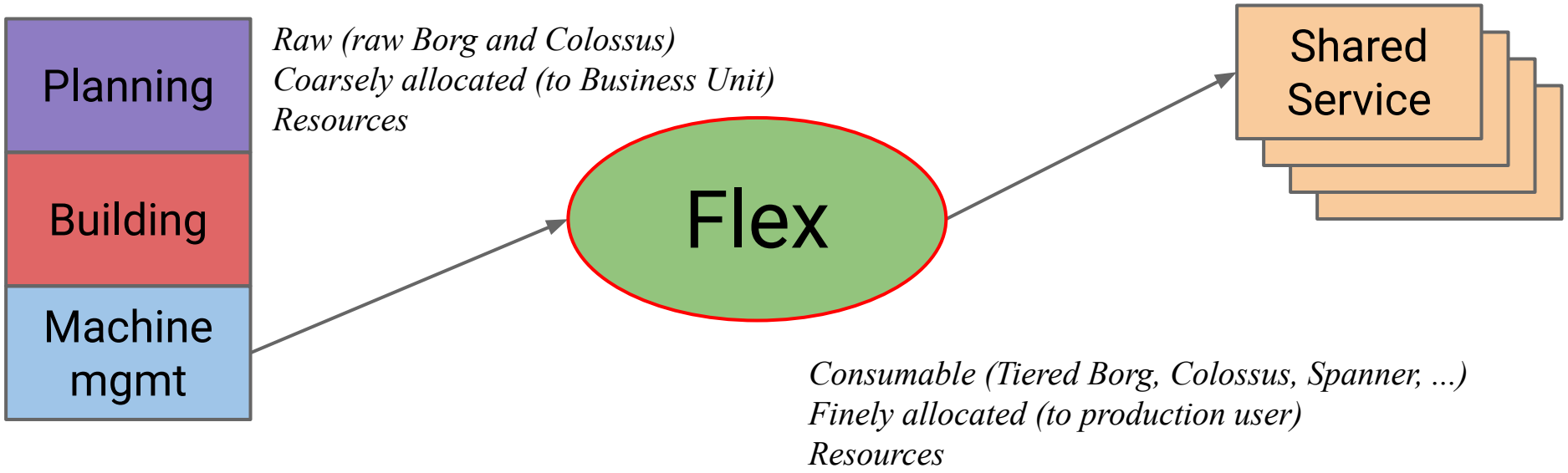
Resource flow



Resource types

- **Physical**: The machines themselves
- **Raw**: Aggregated resources over clusters, which are used to create consumable resources
  - Raw Borg, Raw Colossus (DFS = aggregate disk), Accelerators
- **Consumable**: Resources consumed by the end user
  - Tiered Borg, Tiered Colossus, Spanner, BigTable, ML, ...

Flex supports the cloud operator



Flex value added

- Flex abstracts away the details of having multiple services
  - For planners... who use raw resources
  - For Flex operators... on creating consumable resources via scaling and autoscaling
  - For service users... on obtaining access to different services

- Flex is the perfect place to deploy two efficiency strategies
  - Resource pooling: Resource are allocated on demand, enabling shared safety stock
  - Tiering: consumable (SLO-based) resources enable us to pack more load in the same raw resources

Agenda

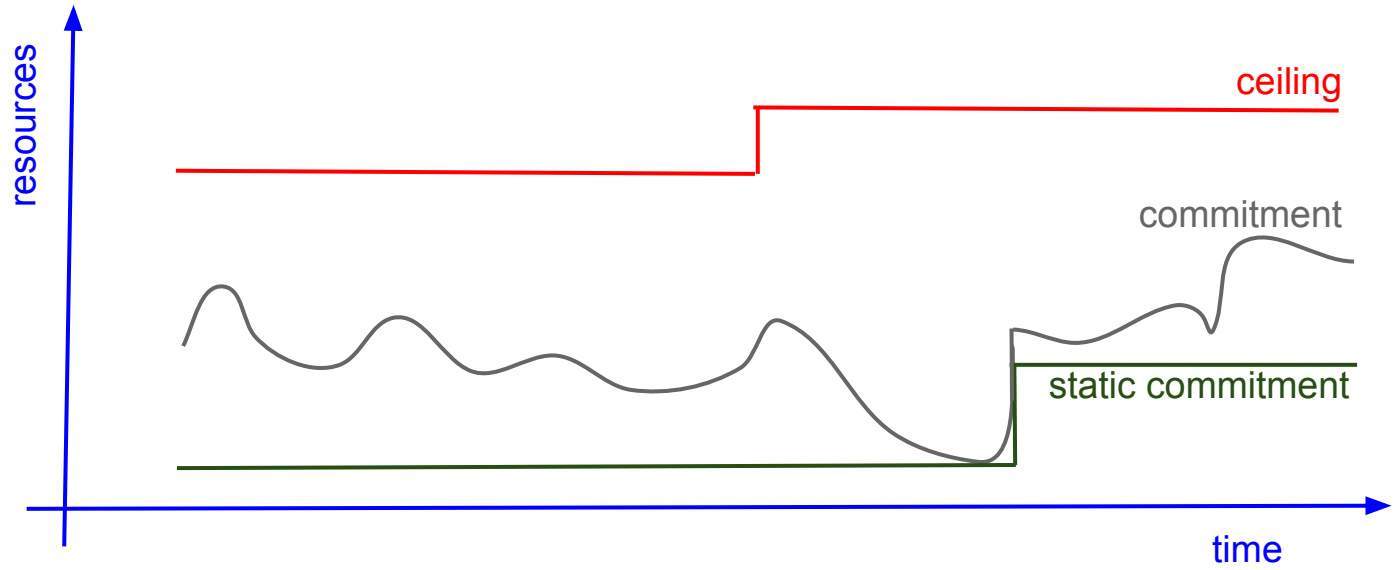
- The Private Cloud Management Problem
- Flex: Google's solution for this problem
  - Flex value proposition
- Resource Distribution
  - Pooling, Scaling
- Tiering and Product Design
- Governance and Obligation

How Flex Works

- Flex is an evolution of the fixed quota model
- Flex automatically and continuously allocates "quota" to cover **usage**
- Flex breaks "quota" into three pieces:
  - **commitment** are the resources really allocated to users at a point in time, it is really "quota" from the viewpoint of the Shared Service
  - **ceiling** is an upper bound to commitment, it is meant to be "quota" from the user viewpoint (which is true if there are no stock-outs)
  - **static commitment** sets a floor to commitment, regardless of usage, guaranteeing they are available even if there is a stock-out

Ceiling, Commitment, Static Commitment

Ceiling and Static Commitment are bounds on Commitment Adjustment



Guaranteeing Resources in Stock-outs

- As static commitment provides a floor for commitments, they allocate resources to a user regardless of these resources being used or not
- **Pros**: Great protection in case of stock out
- **Cons**: Inefficient as fixed quota

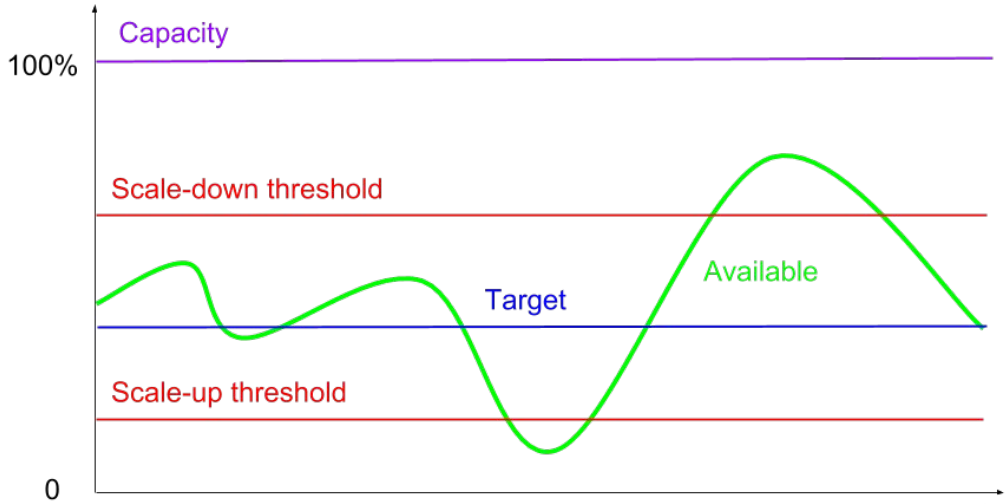
Commitment Adjustment: Applications Already Running Keep Running

- Commitments are used by Flex to tell shared services (Borg, Colossus, Spanner, ...) how much resources are allocated to a user
- Commitments are not oversubscribed
  - That's why shared services often think about commitment as quota

- Commitments grow immediately on resource consumption when user is under ceiling and pool has uncommitted resources
- Commitments decay over time to protect against temporary usage drop, while enabling Flex to reclaim unused allocations

Scaling/Autoscaling for Composite Services

- Create composite resources (Spanner, BT, ...) from foundational resources (Borg, Colossus) in a unified (scaling) and automated (autoscaling) manner
- Automation keeps unallocated base resource fungible
- Key enabler for forecasting and planning to happen for raw resources



Agenda

- The Private Cloud Management Problem
- Flex: Google's solution for this problem
  - Flex value proposition
- Resource Distribution
  - Pooling, Scaling
- Tiering and Product Design
- Governance and Obligation

Tiering != Pooling

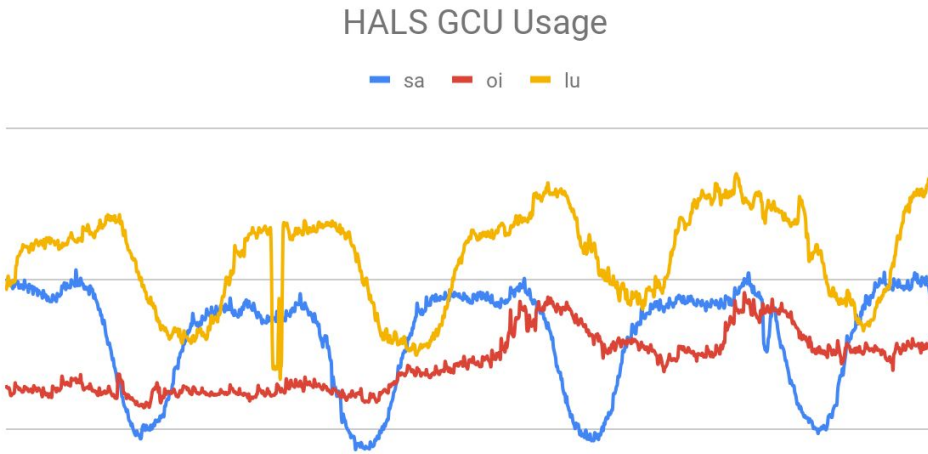
- Flex on-demand commitment allocation allows for pool operators to share the buffer and promise more resources than the pool capacity

  $\sum$ ceiling > $\sum$ commitment

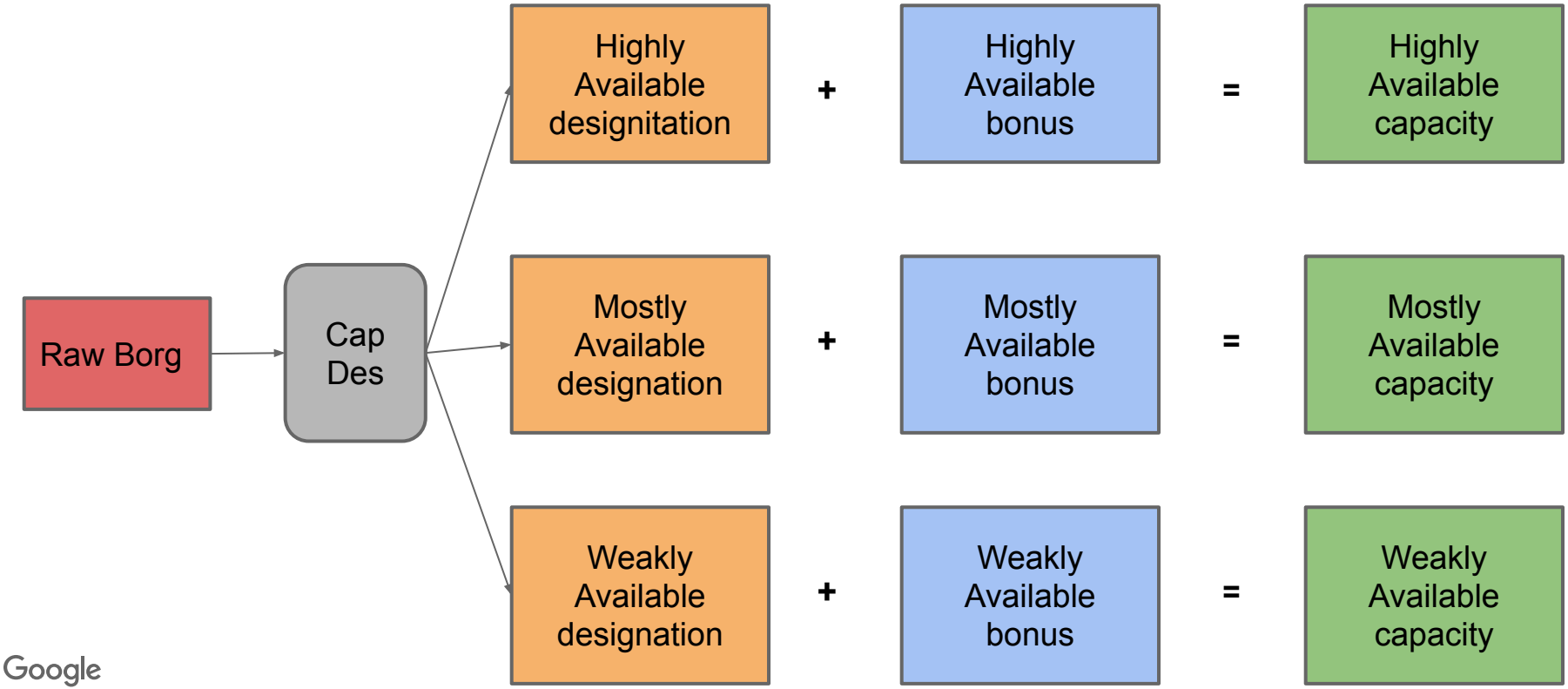
- Tiering is packing more load in the same hardware by designing products that "fill in the cracks"
  - The capacity we make available via Tiering can actually be fully used
  - Tiering is service specific
  - Example for Highly Available Latency Tolerant (HALT):
    Long-term SLOs for reclaimed cloud computing resources. Carvalho, Cirne, Brasileiro, Wilkes. ACM SoCC. 2014

Example: Borg Weakly Available

- Borg Weakly Available offers throughput computing, targeting jobs that need to progress daily, but are flexible on when they run
- We produce WA bonus based on the recurring diurnal cycle
- We are very confident the pattern repeats
- But exactly how much we can fit in the valleys vary from cell to cell and over time
- Therefore, bonus is offered with six month validity

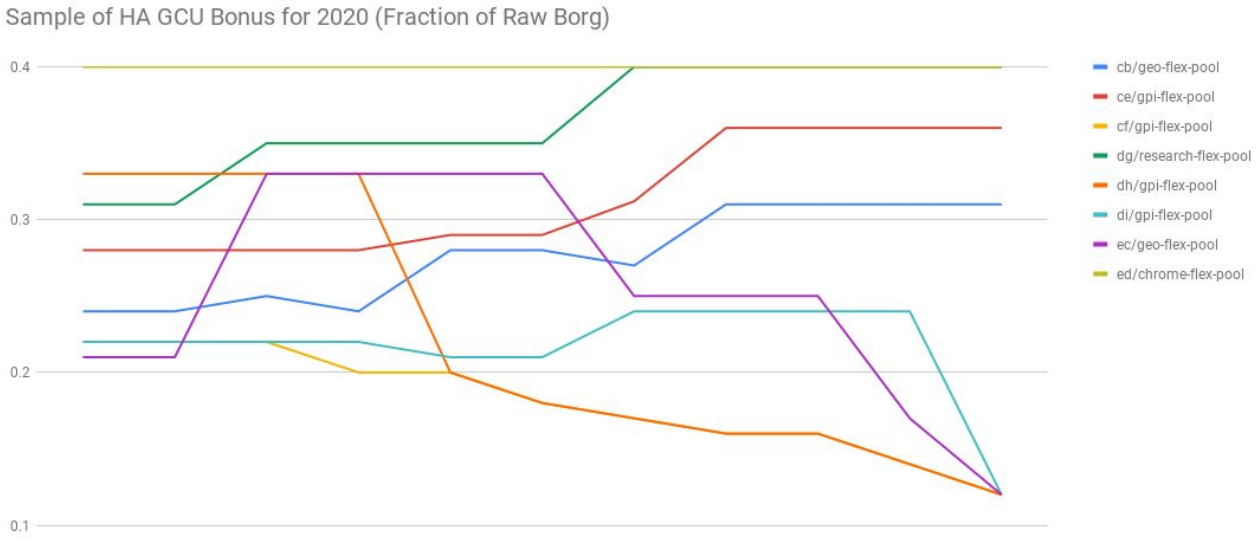


Capacity Designation combines Raw + Bonus



Tiering greatly improves efficiency but bonus makes planning harder

- Fundamentally, bonus makes capacity dependent on the load
- Planners and operators need raw compute to compensate for bonus



* Random sample out of 45K (pool, cell, tier) possibilities

Agenda

- The Private Cloud Management Problem
- Flex: Google's solution for this problem
  - Flex value proposition
- Resource Distribution
  - Pooling, Scaling
- Tiering and Product Design
- Governance and Obligation

Charge Transparency and Oversubscription

- In a private cloud, there is no Profit & Loss
- Instead, we aim for Charge/Revenue Neutrality, i.e. accurately "charge" users for the cost of the resources they consume
  - This is straightforward when hardware is built for a user and not shared with anyone else
  - It is also incredibly inefficient
- Flex promotes resource sharing directly (pooling) and indirectly (tiering)
- How should we charge?
- How to even define charge transparency in Flex?

Obligation: Assigning Financial Blame

- Charge = Obligation * Price
- Obligation captures how much resources a group prompted Google to buy

- For users planned in aggregate (typically small users):
  obligation = commitment * safety_stock_surcharge
- For users planned individually (typically large users):
  obligation = ceiling * oversub_discount

Pricing Tiered Products

- Prices should reflect true costs so that Googlers can make rational choices (aka charge transparency)
- Prices need to be stable to foster good planning
  - The overcommit bonus varies over time because it depends on the load
  - How much raw one needs depends on bonus, and thus on the load

- For tiered services, determining true, stable prices has a circular dependency
  - True cost depends on tier mix, which depends on user behavior, which depends on price, which should equal true cost
- Basing prices only on the existing tier mix may over-incentivise user to user it, which will later raise the price dramatically

Tiered Borg Prices

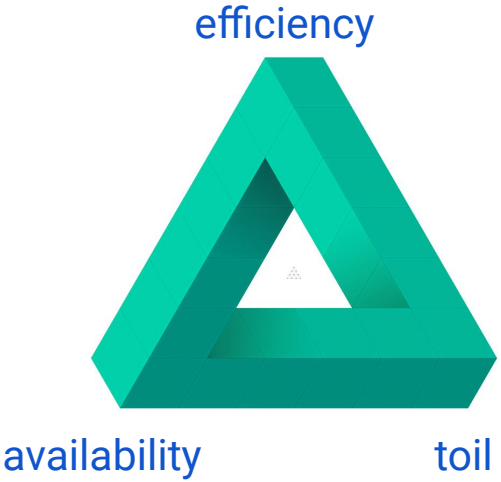
- Tiered Borg prices were initially determined assuming the current global tier mix and bonus, but with a lower bound to prevent over-incentive

	% of Raw
Highly Available	97%
Mostly Available	68%
Weakly Available	47%

- We iterate on prices by computing new prices based on new global tier mix, but we only perform half of the change, to promote stability
- We avoid SLO-inversion by ensuring that better tiers cost more

Summary

- Managing private clouds is a fascinating problem
- We need to balance efficiency, resource obtainability, and user toil to run our cloud efficiently
- We need to do so transparently, as to support proper governance
- We co-designing what services to offer to achieve these goals



Thanks!