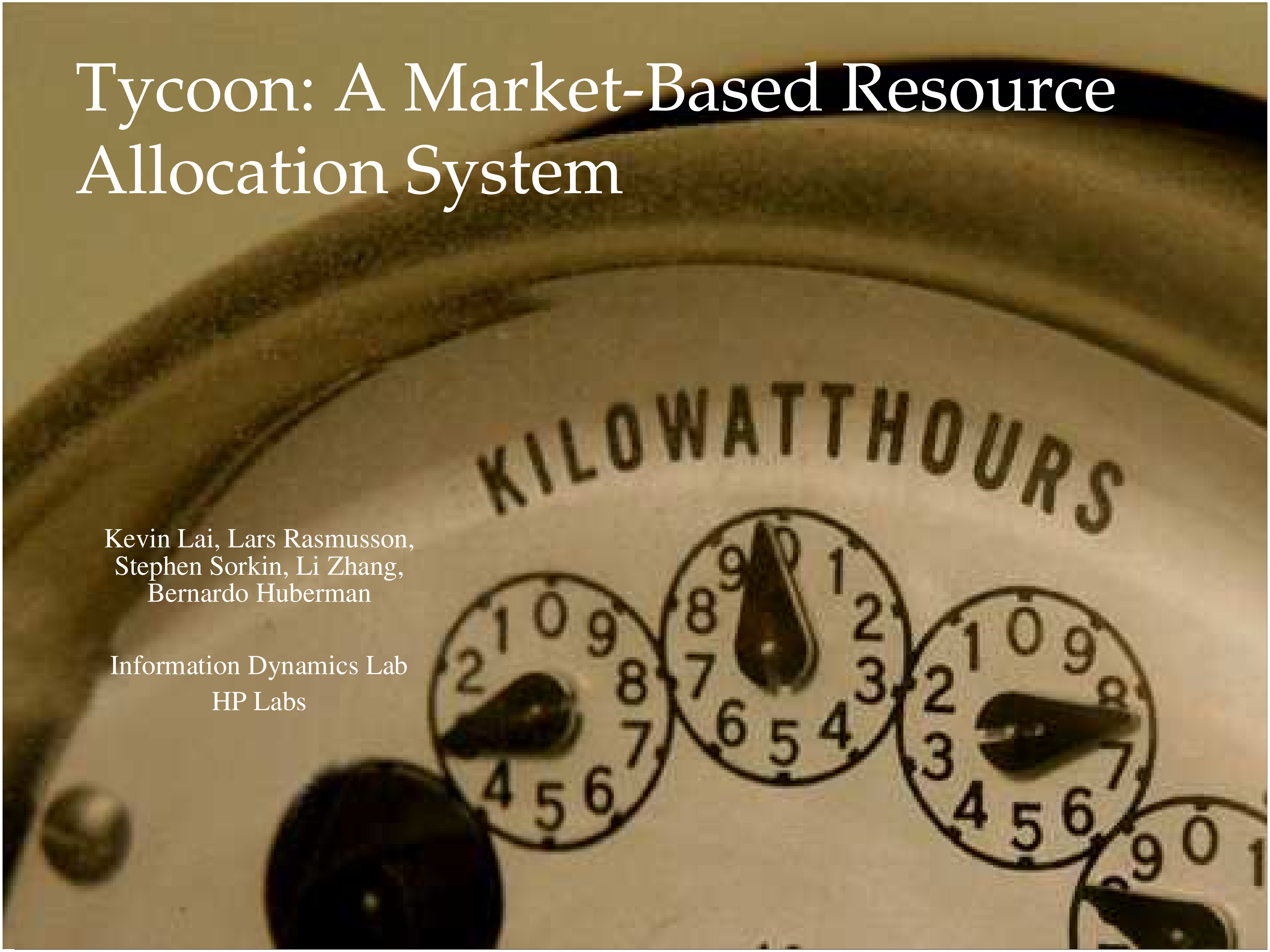
Tycoon: A Market-Based Resource Allocation System



Kevin Lai, Lars Rasmusson,
Stephen Sorkin, Li Zhang,
Bernardo Huberman

Information Dynamics Lab
HP Labs

Motivation

- Distributed shared clusters
  - Grid, PlanetLab, the internal clusters of companies
- Applications:
  - scientific applications, databases, web servers, email servers, etc.
- Sharing distributed computers potentially
  - ***increases throughput*** (statistical multiplexing)
  - ***lowers delay*** (geographic dispersion)
  - ***increases reliability*** (redundancy in hosts, network connections, etc.)

Problem

- Currently, shared resources (CPU cycles, disk, etc.) are
  - **Poorly utilized** (not given to the most important task)
  - **Slow to adapt** (adapt = reallocate resources)
  - **Expensive to manage** (in user time)

Tycoon

- market-based system for resource allocation
  - *distributed* markets allocate local resources
  - users bid *continuously* for *virtualized*, *proportional* resources
  - users only pay for resources consumed
- low overhead, low latency markets
  - agility: can shift all resources in system in < 10 seconds
  - scalability: current platform scales to (active users)(hosts) = 12,000
- arbitrarily more efficient utilization than Proportional Share
  - more efficient even when users do not actively bid
- removes need for users to negotiate resource allocation

Common Non-Economic Approaches

- over-provision
  - expensive, complementary solution
- manual allocation
  - time-consuming and/or inefficient to manage more than 100's of machines, 10's of active users
- scheduling
  - assumes truthful task valuation
  - produces optimal offline schedule using NP-hard algorithm
  - online algorithms using heuristics are not optimal
- Proportional Share

Proportional Share

- Administrator sets weights, e.g., $w_{alice} = 2 \quad w_{bob} = 1$
- System with r resources allocates to user i a share of

  $r \cdot \frac{w_i}{\Sigma w_i}$ e.g., Alice gets 2r/3, Bob gets r/3

- *Economically Inefficient*
  - no incentive to truthfully differentiate importance of jobs
- *Slow to adapt*
  - changing weights requires involvement of administrator
- *And/or expensive*
  - Alice and Bob negotiate (communication costs of n^2)
- Easy to use
  - run whenever you want, no bidding required

Economic Related Work

- Auction
  - method for accurately determining value of something
  - explicitly assumes strategic behavior
  - opens: bidding starts
  - closes: bidding stops, resource assigned to winner
  - different forms induce different bidding behavior
- frequency of auction
  - infrequent
    - high delay between wanting a resource and close $\rightarrow$ poor agility, ease-of-use
    - speculation: early winner can sit on resource denying it to a later user who values it more
  - frequent: can't hold a resource for very long $\rightarrow$ poor predictability

Auction Issues, continued

- delay between auction close and resource use
  - long: poor agility, ease-of-use
  - short: poor predictability
- winner's curse
  - user wins auction, does not want resource at clearing price
  - difficult to accurately predict application resource consumption
    - deterministic workload: e.g., given scene to render, variance of estimate is ~50%
    - non-deterministic workload: extremely difficult
- Auctions require significant modifications to be used in a resource allocation context

Outline

- Service Model
- Interface
- Architecture
  - Auctioneer
  - Agent
- Experiments
  - Agility
  - Overhead

Service Model

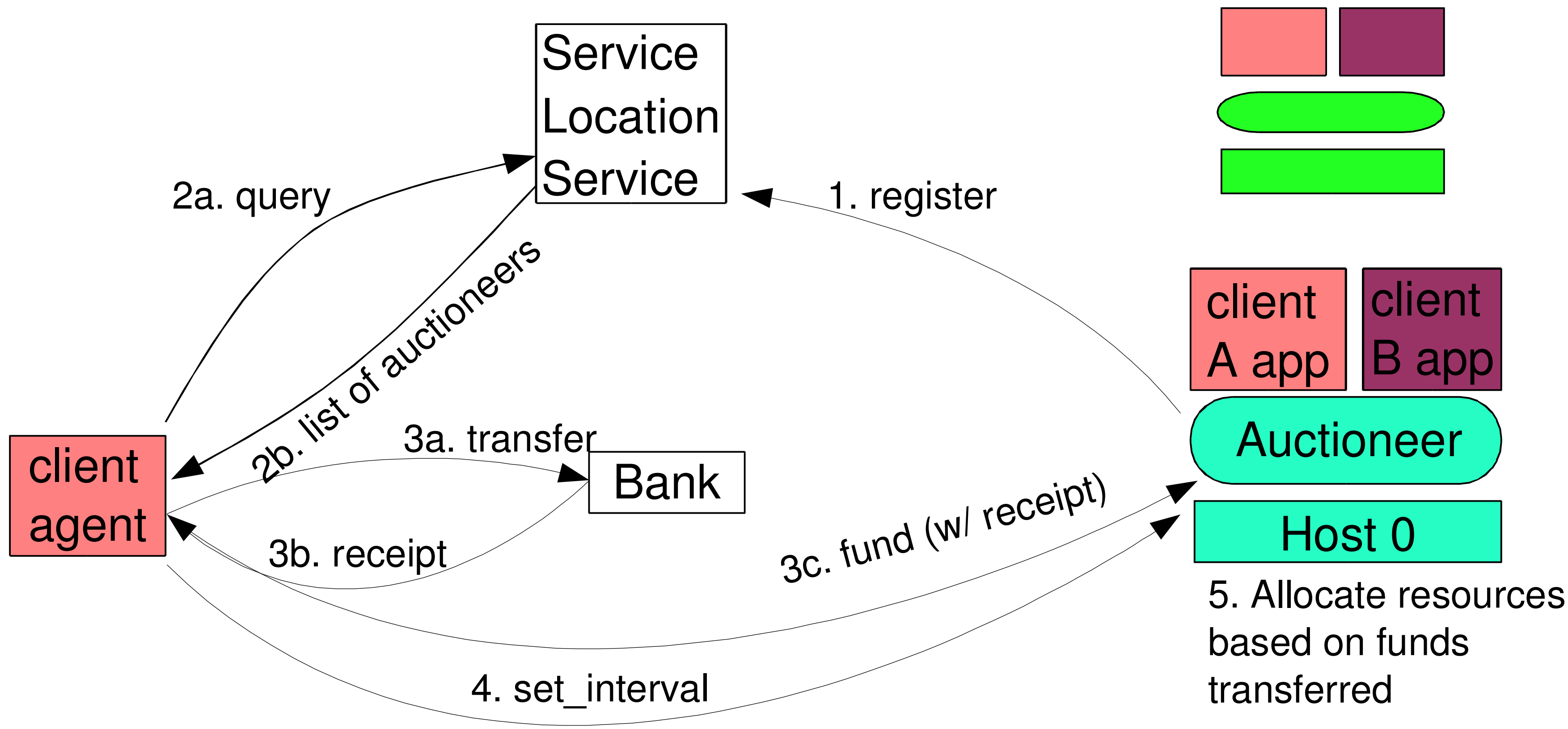
- Users have a limited budget of credits
- Users bid for resources
  - bid = (h, i, e, b, t)
  - h: host, i : user, e: resource type, b: amount of credits, t: bidding interval in seconds
  - *continuous* bid
  - ssh into host to use resources
- auctioneer on h allocates resources
  - in proportion to user i's weight = b_i^e / t_i^e
  - independently of other auctioneers
  - only charges users for resources consumed
  - cost of resources can change at any time

Prototype User Interface

- Create an account on a host
  - `tycoon create_account host0 10`
- Run
  - `ssh klai@host0 my_program`

- Optionally:
  - Transfer more credits into account
    - `tycoon fund host0 cpu 10 1000`
  - Change bidding interval
    - `tycoon set_interval host0 cpu 2000`
  - Determine current balance, resources allocated, etc.
    - `tycoon get_status host0`

Architecture

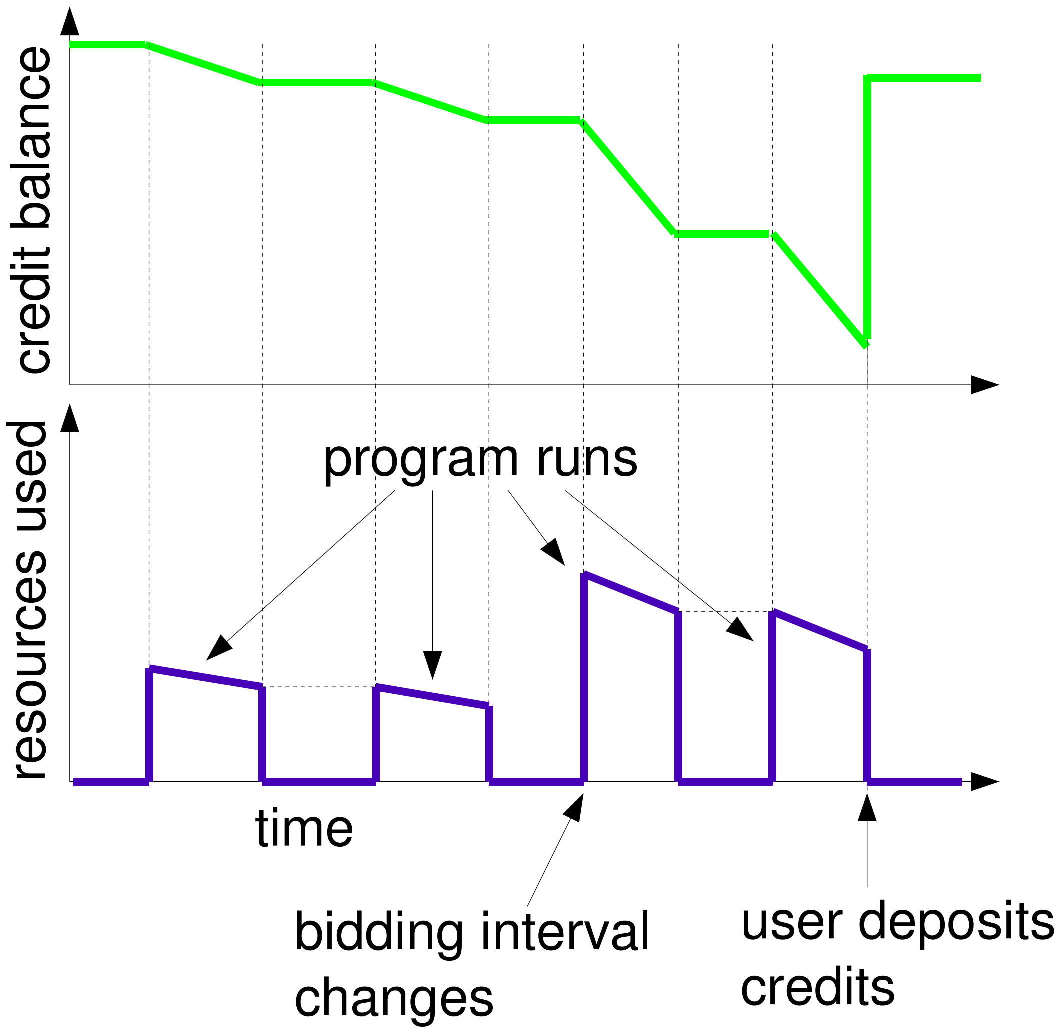


- Hosts do independent allocation
- 3 is relatively expensive, 4 is less expensive alternative

Auctioneer: Allocating Resources

- bid = (h: host, i : user, e: resource type, b: amount of credits, t: bidding interval)
- R^e : total amount of resource e, q_i^e : amount of e used by user i per second
- auctioneer on h allocates resources
  - user i's weight: b_i^e / t_i^e
  - amount of e allocated to user i per second: $r_i^e = \frac{b_i^e / t_i^e}{(\Sigma\, b^e / t^e)} R^e$
  - amount user i pays per second: $s_i^e = min\,(\frac{q_i^e}{r_i^e}, 1)\frac{b_i^e}{t_i^e}$
  - bid is automatically recomputed: $b_i^e = b_i^e - s_i^e$
  - currently recomputed every 10s → mean 5s to reallocate
  - only charged for resources used → don't have to withdraw bids
  - credits last a very long time → don't have to update bids

Using Continuous Bids



- separation of credit amount from bid interval allows user to control frequency of deposits
  - less interaction required
  - less load on bank

Client Agent: Distributed Bidding

- Manual bidding in 1000's of markets is not practical
  - Resources available on hosts varies
  - Demand for resources on hosts varies
  - ideally user just specifies a total budget of X
- simple algorithms can be far from optimal
- Best Response Algorithm
  - user i has a preference $p_i^e(j)$ for resource e on host j
  - $x_i^e(j)$ is the amount bid by user i for resource e on host j
  - $y_i^e(j)$ is the amount bid by all users except i for resource e on host j
  - maximize $\sum_{j=1}^{n} p_i^e(j) \frac{x_i^e(j)}{x_i^e(j)+y_i^e(j)}$ s.t. $\sum_{j=1}^{n} x_i^e(j) = X$
  - use Lagrangian multipliers

Best Response Algorithm

- Requires O(n log n) time
- results in multiple Nash equilibria
  - some have very low economic efficiency
- preliminary simulation shows that its mean efficiency is ~90%
  - simulation details requires a separate talk

Verification

- potential auction pitfall: auctioneer cheats
- possible solutions
  - trusted computing platform
  - audit log
- Tycoon solution
  - substitute application-layer cost-effectiveness metric for preference instead of generic resource
    - e.g., (frames rendered / s) / credit instead of CPU cycles / s
  - best response algorithm will automatically favor hosts that have a high application cost-effectiveness
    - hosts that have a poor (frames rendered / s) / credit will get dropped
  - treats cheaters as hosts with poor cost-effectiveness
  - reduced spending by agents → reduced incentive to cheat

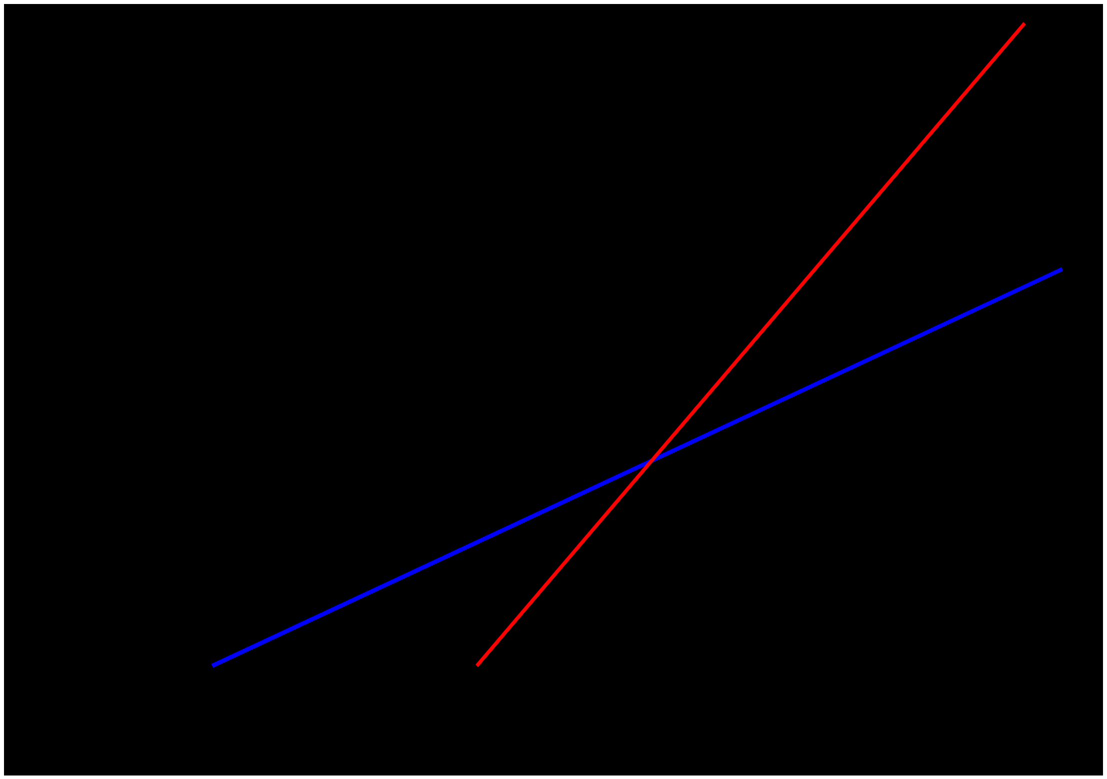
Experiments

- Prototype implementation
  - only manages CPU cycles because of limitations in VServer
- Runs on 20 hosts
  - 8 in Bristol, U.K.
  - 450 Mhz - 1 Ghz x86
  - RedHat Linux 9.0

Agility

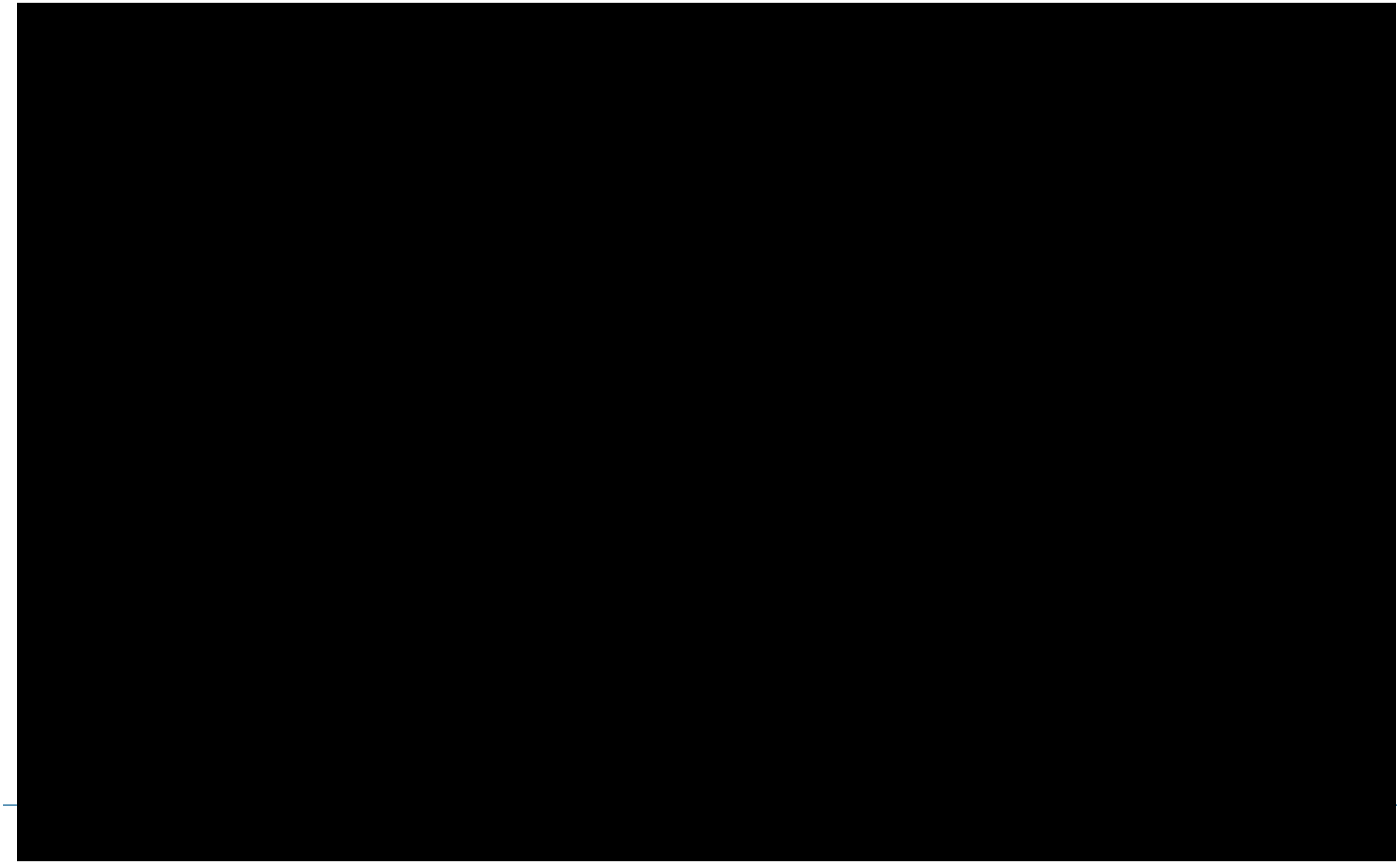
- progress of a scene being rendered on cluster using Maya 6.0
- frames are distributed to different hosts in cluster
- user changes bid by changing bidding interval on all hosts at 185s

Agility

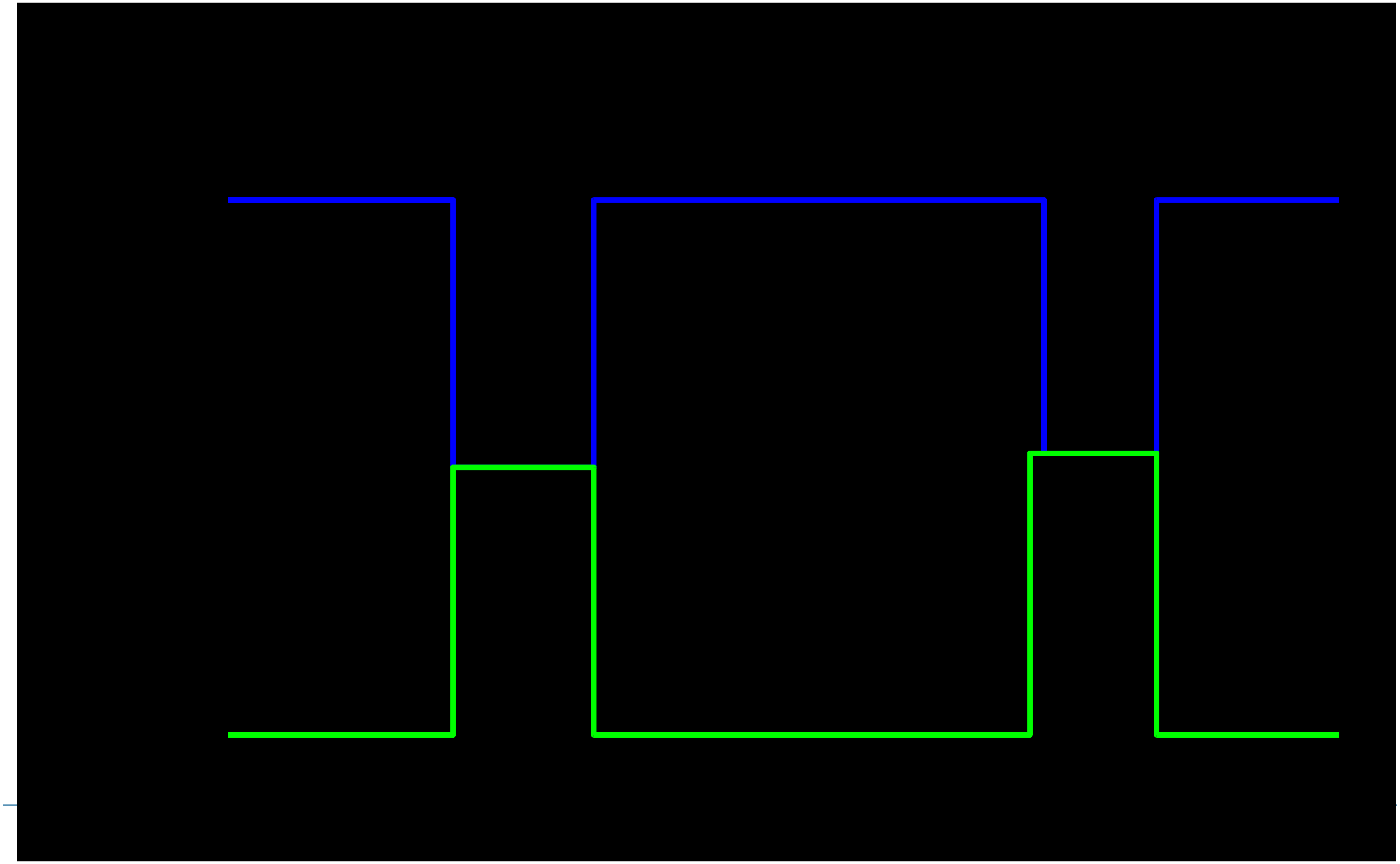


- hosts begin reallocating in < 10s
- last bid change finishes at 211s
  - limited by client host, application structure
- agility key for unpredictable server applications
  - 3-tier ecommerce
  - media serving
  - web, email, etc.

Compared to Proportional Share



Compared to Proportional Share



Overhead

- VServer overhead
  - CPU bound process: ~3%
  - system call-heavy process: ~10%
- Protocol overhead
  - one centralized Service Location Service with 100Mb/s Ethernet supports at most 75,000 hosts
  - one centralized 450MHz bank supports (active users)(hosts per user) = 12,000
    - e.g., 24 active users, 500 hosts per user
    - assumes users deposit funds every 20 minutes
    - limiting operation is DSA public key authentication
    - protocol could be optimized to include several deposits in one message
    - centralized bank is not likely limit scalability in practice

Miscellaneous Topics

- Virtualization
  - Linux VServers + PlanetLab plkmod
- Security protocols
  - all messages are signed + nonces
- Predictability of resources
  - agents can reserve credits to be used in case prices rise
- Scalable communications with auctioneers
  - can use application-layer multicast to distribute bids to auctioneers
- Multiple resources
  - auctioneer periodically re-balances separate credit reservoirs for each resource
- Different allocation algorithms
  - future work

Summary

- continuous bids
  - easy to use
    - don't need to plan ahead
    - don't need to update
  - computationally efficient
  - low latency to change allocation
- distributed markets
  - agile: only manage local resources
  - fault-tolerant