PLATFORMLAB

COLLIN LEE

INITIAL DESIGN THOUGHTS FOR A GRANULAR COMPUTING PLATFORM

GOAL OF THIS TALK

- Introduce design ideas and issues for a granular computing platform.
- Stimulate discussion.
- Straw-man ideas and opinions up for debate.
  - Structured as a collection of “what-if’s” and assertions.
  - Ultimately looking for safe simplifying assumptions.
- Encourage feedback and suggestions.
  - Feel free to point out the good, the bad, and the ugly.

OVERVIEW

- What is granular computing?
  - Marching forward with or without applications.
  - Anatomy of a Granular Application.
  - Properties of a Granular Application.
- Assertions:
  - All code needs to be pre-loaded and ready to run.
  - A shared polling infrastructure is needed.
  - Trade memory protection for low latency.
  - Scheduling must be decentralized.
  - Fast resource preemption is necessary.
  - Granules can only communicate via invocation.
  - Durability of granules must be batched.
  - Need reliable networks for low latency (geo-)replication.
  - Storage system must expose data location.
- Conclusion
- Questions and Feedback
- Poster Session

WHAT IS GRANULAR COMPUTING?

LARGE NUMBER OF TINY TASKS

RAPID SCALE UP/ DOWN

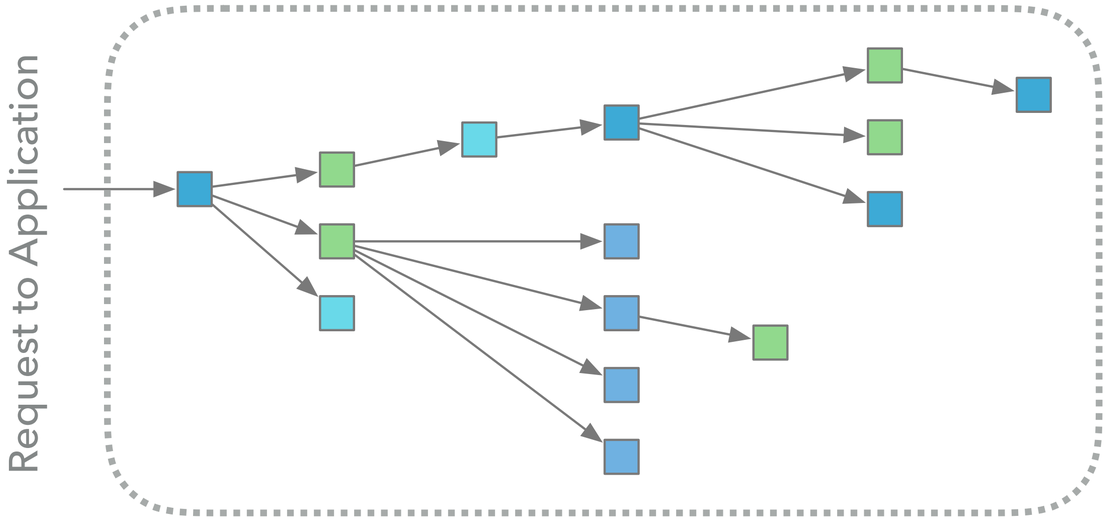
APPS COMPOSED OF TINY TASKS

MARCHING FORWARD WITH OR WITHOUT APPLICATIONS

- Current lack of concrete applications.
- Possible approach: wait for applications?
  - Have yet to find one.
  - Applications and platforms might be “chicken and egg.”
- Current approach: assume application properties and proceed.
  - Risk: we could be totally wrong.
  - Belief: we will learn a lot anyway.
  - Results can be used as input for designing future applications and platforms.
- If you do have applications, let us know!

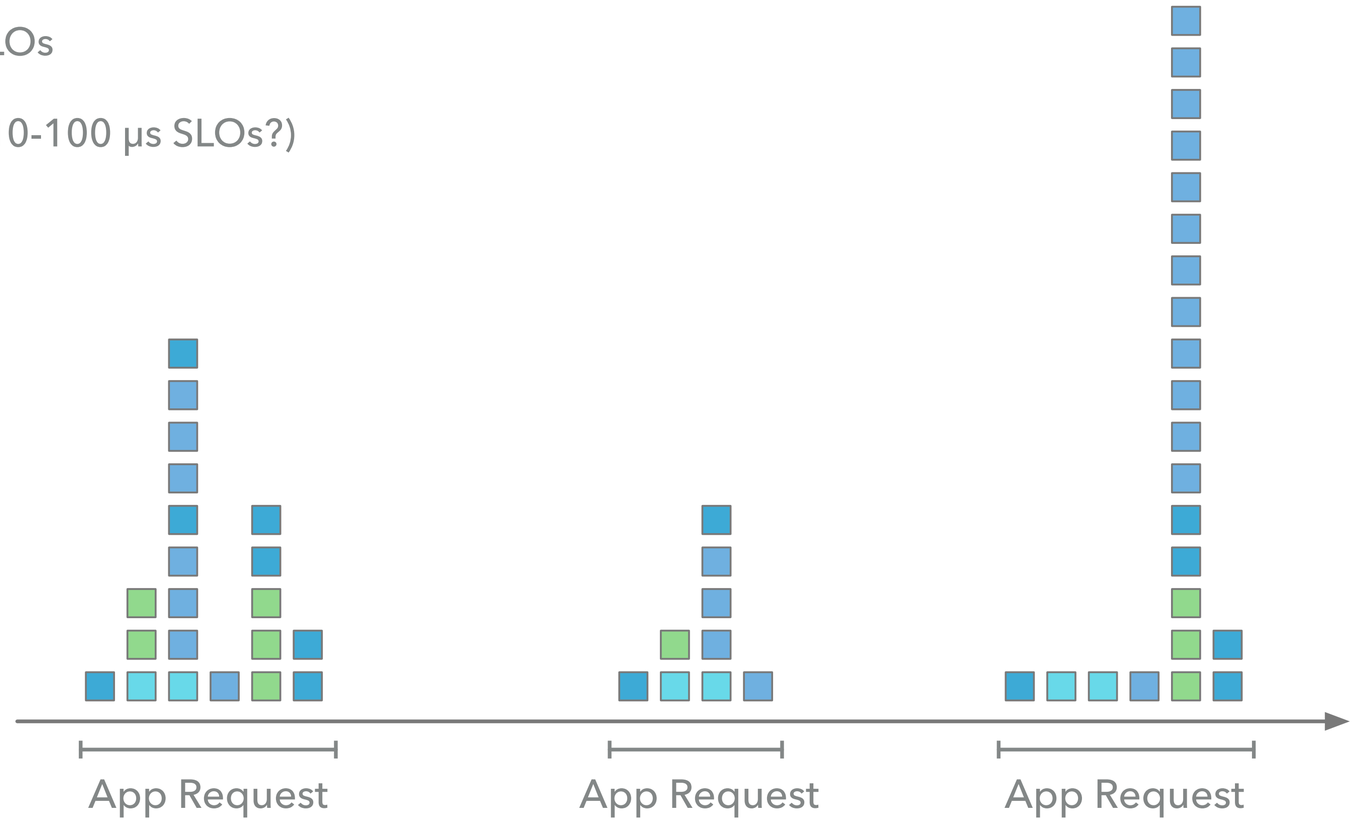
ANATOMY OF A GRANULAR APPLICATION

- Application
  - Composed of a collection of granule types.
  - e.g. Backyard surveillance
- Granule type
  - Distinct unit of execution.
  - e.g. Detect Cat, Detect Mountain Lion
- Request to application invokes a large number of granules.
  - Invocations both in parallel and in series.



PROPERTIES OF A GRANULAR APPLICATION

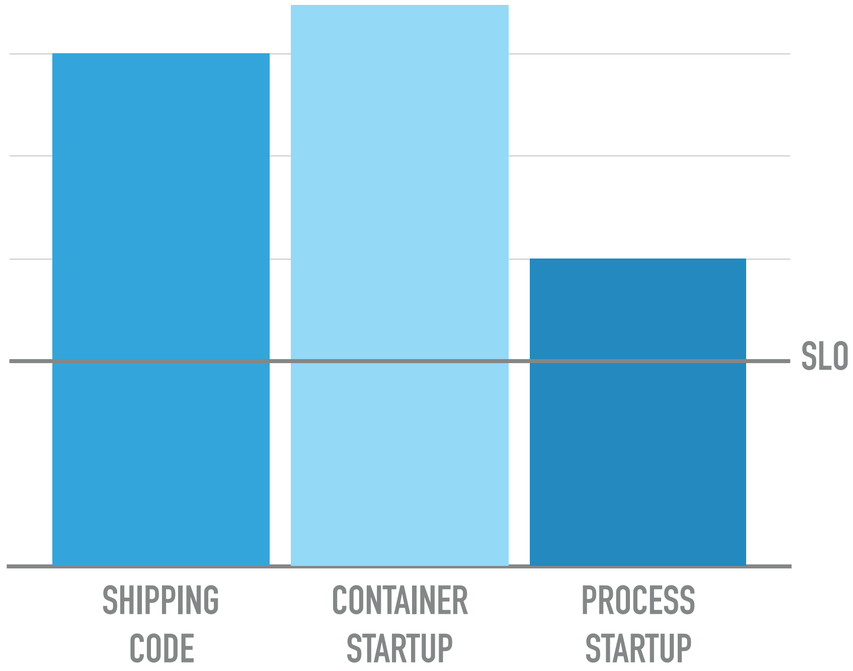
- Application vs Granule Latency
  - Application Requests Latency: 1-10 millisecond SLOs
  - Granule Execution Latency: As small as possible (10-100 µs SLOs?)
- Low Latency Granules
- Low Duty Cycle
  - Bursty: scales to full cluster in short bursts
  - Dynamic workload
- Unpredictable Execution Paths
  - Dynamic/Data dependent code paths
  - Unpredictable granule execution



ASSERTIONS . . .

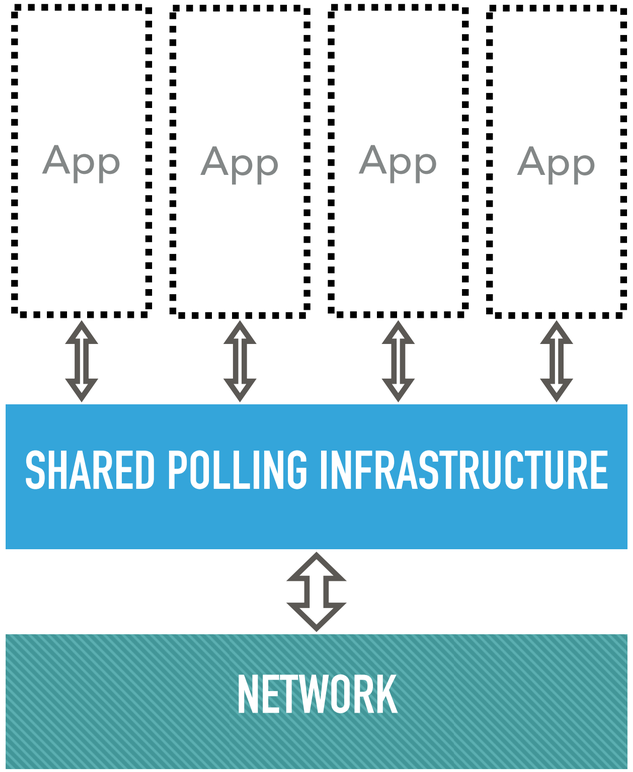
ALL CODE NEEDS TO BE PRE-LOADED AND READY TO RUN

- Cost of dynamic code load is high.
  - Remote code shipping (100+ ms), Container startup (300-400 ms), Process startup (~1 ms).
- Dominates 10-100µs granule SLO.
- Solution: Have code already loaded and ready to execute.
  - Burst request needs code to be on every machine.
  - Every machine must be ready to run (~)any granule.
  - New payment model: pay for hot loaded code.
- Questions:
  - Does all the code in a datacenter fit in the memory of a single machine?
  - Would the new Non-Volatile Memory technology help?



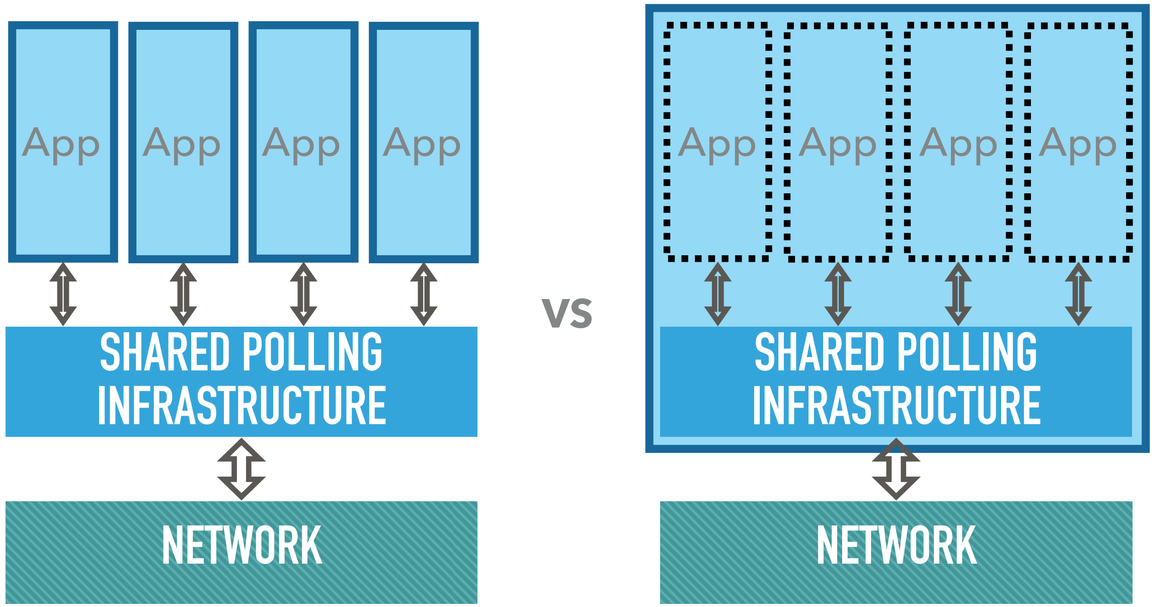
A SHARED POLLING INFRASTRUCTURE IS NEEDED

- Need polling for low latency invocation.
  - Experience from RAMCloud shows that interrupts and signals are too slow (~500µs).
- Large number of polling applications would overwhelm a machine.
  - Might have 100s of applications and 1000s of granule types.
  - Applications need to poll for lowest invocation latency.
- Solution: Shared polling for all applications on a machine.
  - Reduces idle resource usage to single poller.
  - Dispatches granule invocation requests for execution.
- Questions:
  - Is polling necessary? Are there better ways to dispatch work without polling?
  - How does a shared poller work?



TRADE MEMORY PROTECTION FOR LOW LATENCY

- Process switch cost is too high for granule invocation.
  - Suppose a process per application (or granule type).
  - Shared polling would need to signal between process.
    - Signal Cost (~5µs), Process Switch Cost (~0.5ms).
- Solution: Run all applications under a single process.
  - Applications are shared libraries.
  - Dispatch can be fast user-thread creation (< 200ns).
  - Lose memory protection between applications.
- Questions:
  - Is this an acceptable trade off?
  - Are there other ways to provide memory protection?



SCHEDULING MUST BE DECENTRALIZED

- A full cluster might see 1 to 100 billion granules per second.
  - 1 billion granules per second (100µs granules on 100,000 cores).
  - 100 billion granules per second (10µs granules on 1,000,000 cores).
- Too much to schedule in a centralized manner.
- Solution: Decentralized scheduler with minimal information.
  - Levels for scheduling:
    - Local: schedule on the invoking machine (<200ns).
    - Remote random: send an RPC (3-5 µs - 1/2 RTT).
    - Remote load balanced: power of two choices with RPCs (8-15µs - 1.5 RTTs).
  - Simplified by property of low duty cycle.
- Questions:
  - How much load balancing do we really need?
  - How much will you pay in latency for load balancing? (Good balancing costs latency)

FAST RESOURCE PREEMPTION IS NECESSARY (1)

- Handling application bursts require provisioning resources for peak load.
  - Without sufficient resources, low latency during high load is not possible.
- Busty applications are by definition low duty cycle.
- Low duty cycle applications leave significant idle resources during low load.
- Idle resources can be used for low priority jobs (improving utilization).
  - Aggressive use of idle resources increases utilization.
  - e.g. batch jobs, high latency SLO jobs, etc.
- Granule invocation may require preempting execution of low priority jobs.
  - If a low priority job is running when a burst occurs, the job may need to get out of the way.
- Invoking a granule may incur the cost of preempting low priority jobs.
  - Aggressive idle resource usage means that a granule invocation request preemption.
  - No known preemption mechanism with cost under 10s of µs. (best known is Arachne @ 22µs)

FAST RESOURCE PREEMPTION IS NECESSARY (2)

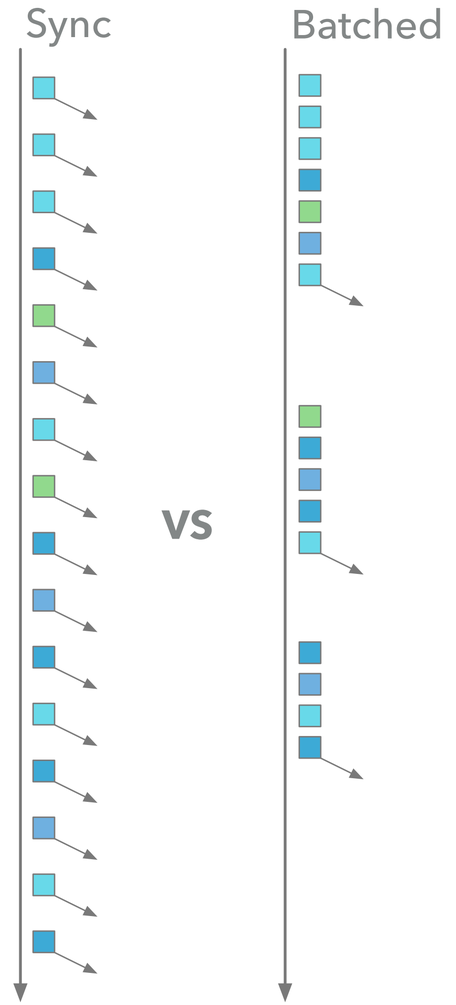
- Can only have 2 of 3:
  - Low latency
  - High Resource Utilization
  - Slow Preemption
- Solution 0: Give up on high utilization
  - Leave more resources idle.
  - Hope that we can detect a burst early enough to mass preempt resources.
- Solution 1: Build a faster preemption mechanism.
- Questions:
  - Are there preemption mechanisms we are missing?
  - Can we amortize the cost of preemption across a burst?

GRANULES CAN ONLY COMMUNICATE VIA INVOCATION

- Granule execution times are too short for significant communication.
  - A granule might only execute for 10 - 100 µs.
  - Not enough time to find the other granule (which could be anywhere) and communicate.
- Granules must have some communication
  - Otherwise, the granule model may be too restrictive?
- Solution: Communicate during invocation
  - Communicated information is passed along with the invocation.
  - No need to “find” the target granule.
- Questions:
  - Can we still write interesting applications where all communication is done via invocation?

DURABILITY OF GRANULES MUST BE BATCHED

- Making computation results durable means a write to a storage system.
  - RAMCloud can do small writes in 100B in 15μs, larger 1kB writes in 17μs.
- Making each individual Granule durable would be too slow.
  - Doing storage system write could double the latency of a granule.
- Solution: Defer durability of a related set of granules and batch the “write.”
  - Allows amortization of write overheads
  - Removes durability cost for intermediate results.
  - Creates set of granules that will externalize their results together and rerun in case of failure.
- Questions:
  - How do we define the set of granules to be batched?



NEED RELIABLE NETWORKS FOR LOW LATENCY (GEO-)REPLICATION

- Replication traditionally requires synchronous updates to remote storage.
  - Intra-datacenter replication in µs, inter-datacenter (geo) replication in ms.
  - Replication is needed for durability.
- Synchronous replication (especially geo replication) may be too slow.
  - Granular computing has granules executing in µs and applications in low ms.
- Solution: Rely on replication of messages over a reliable network.
  - Assume “lifeboats away” model of replication.
  - e.g. If 3 replication requests are sent on the network, the network should guarantee delivery of at least 1 replication request.
- Questions:
  - Can we trust networks to be reliable?

STORAGE SYSTEM MUST EXPOSE DATA LOCATION

- Short granules can’t spend too much time collecting data.
  - Remote read costs > 5µs.
  - 10 - 100 µs granule execution may not allow (that much) time for remote data reads.
- Colocating a granule with its working dataset can improve performance.
- Solution: Have storage systems expose information about data location.
  - Scheduler can use this information to schedule granules closer to the data.
  - Requires granules to expose information about their working dataset.
- Questions:
  - For this to work, most storage systems need to provide location info. Is there any easy way for storage system to do so?
  - Is it possible for granules to know their data dependencies on invocation?

CONCLUSION

- What is a granular computing platform?
  - **Platform** for applications composed of a **large number of very small tasks** that **rapidly scale up/down**.
  - 10 – 100µs granules
  - Bursty workloads
  - Dynamic code paths
- Goal of talk: stimulate discussion.
- Assertions:
  - All code needs to be pre-loaded and ready to run.
  - A shared polling infrastructure is needed.
  - Trade memory protection for low latency.
  - Scheduling must be decentralized.
  - Fast resource preemption is necessary.
  - Granules can only communicate via invocation.
  - Durability of granules must be batched.
  - Need reliable networks for low latency (geo-)replication.
  - Storage system must expose data location.
- All feedback welcome.
- Poster session later today.

PLATFORMLAB

THANK YOU

QUESTIONS AND FEEDBACK