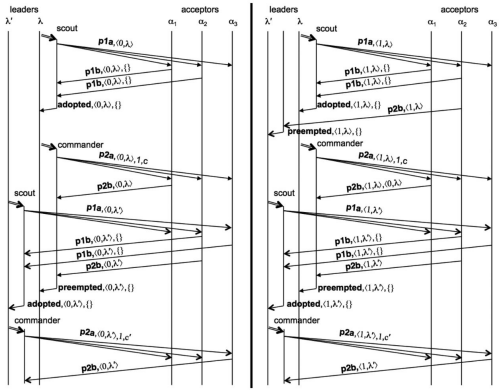
LambdaObjects:
Re-aggregating Storage and Execution for Cloud Computing

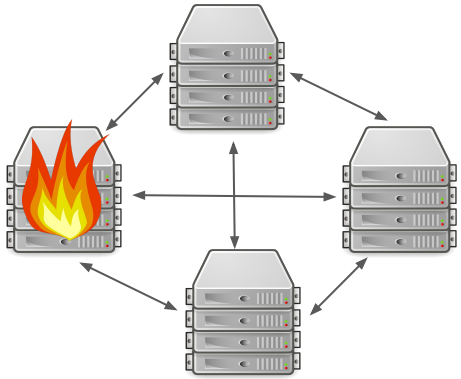
Kai Mast, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau
University of Wisconsin-Madison

Challenges when Developing Scalable Applications



[Van Renesse; 2015]

Concurrent systems are hard to reason about



Hardware failures are common

Workloads might change unexpectedly

Developers want better abstractions for **elastic** and **scalable** applications

Serverless Programming So Far

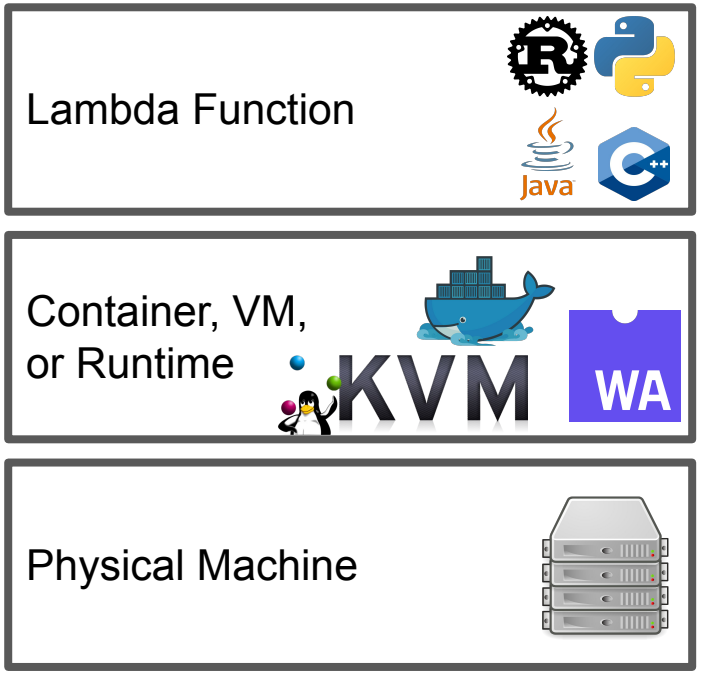
Split application into multiple **lambda functions** deployed on a cloud service

Advantages:

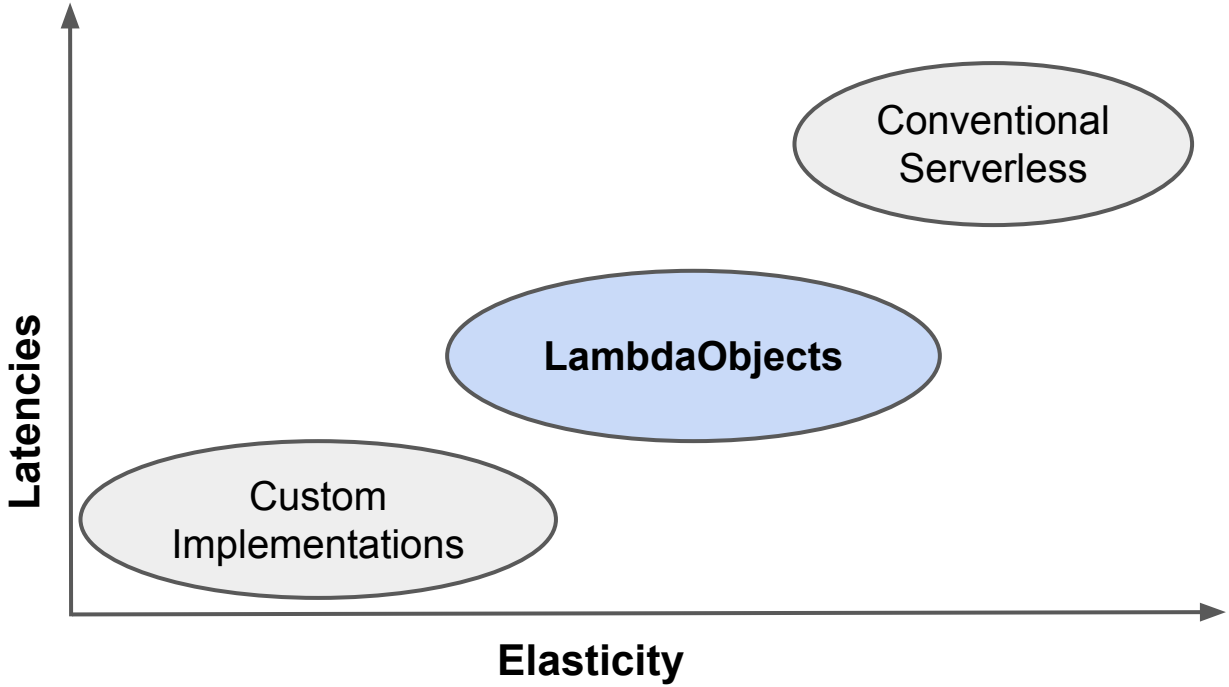
- Hides the underlying distributed system from the application developer
- Fully **elastic**: Resources are automatically allocated by the cloud provider

Limitations:

- High Latencies
- Weak Consistency Guarantees

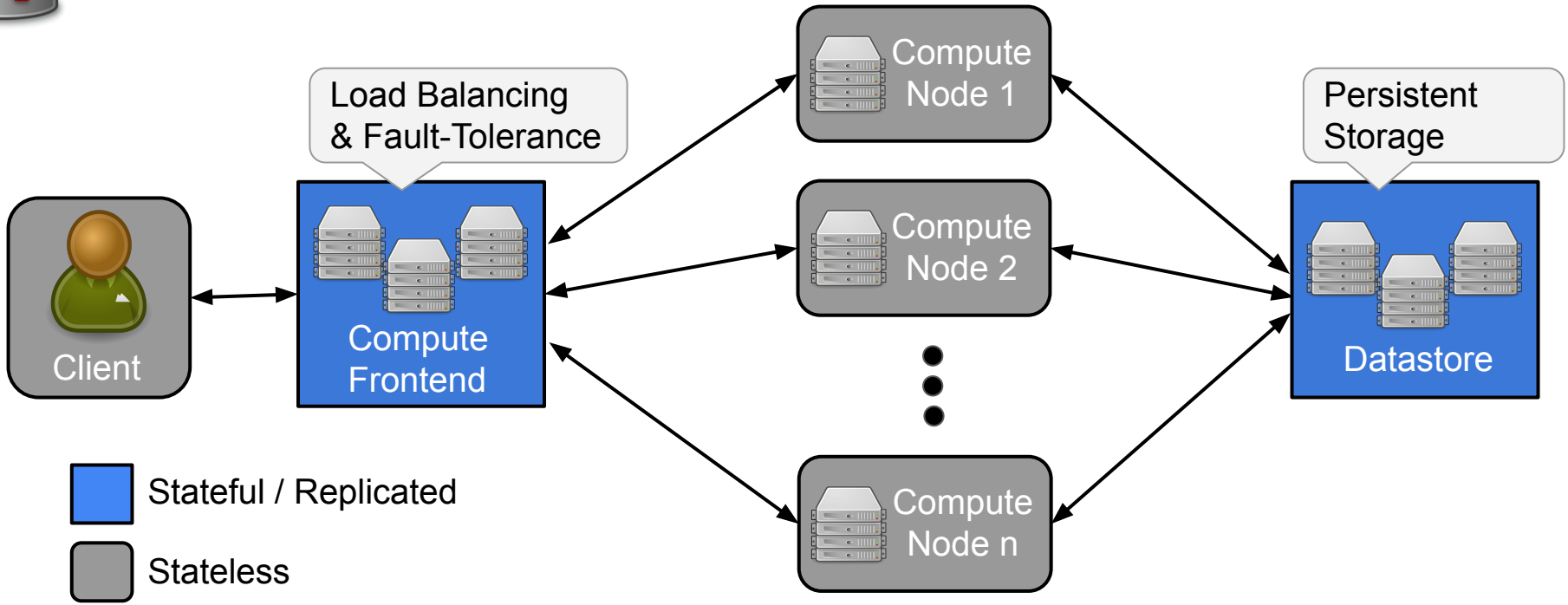


Towards a Better Trade-Off



Goal: Keep latencies low enough to be unnoticable for the end user, but maximize elasticity.

Disaggregation in Serverless



High latencies due to **replication of work** and **lack of locality**

Re-Aggregation: Challenges

- Determine which data and computation belongs together
- Design a compute-enabled datastore
- Support mutually distrusting functions
- Adapt to workload changes
- Provide transactional guarantees

An Abstraction for Re-Aggregation

LambdaObjects bundle data and functions that **logically belong together**, similar to classes in object-oriented programming

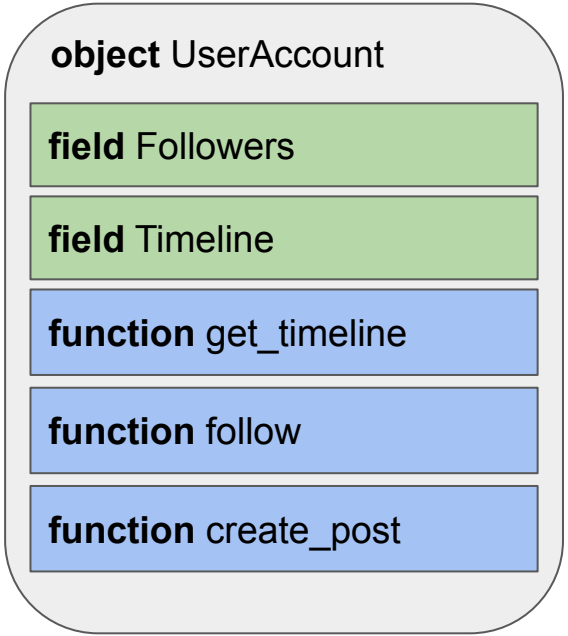
Avoids expensive data transfers:

- All data of an object resides on the same machine (or replica set)
- Function invocations of an object execute at the machine holding its data

Enables strong consistency:

- No cache layers in place
- Easy to avoid scheduling functions modifying the same object

LambdaObjects: Application Example

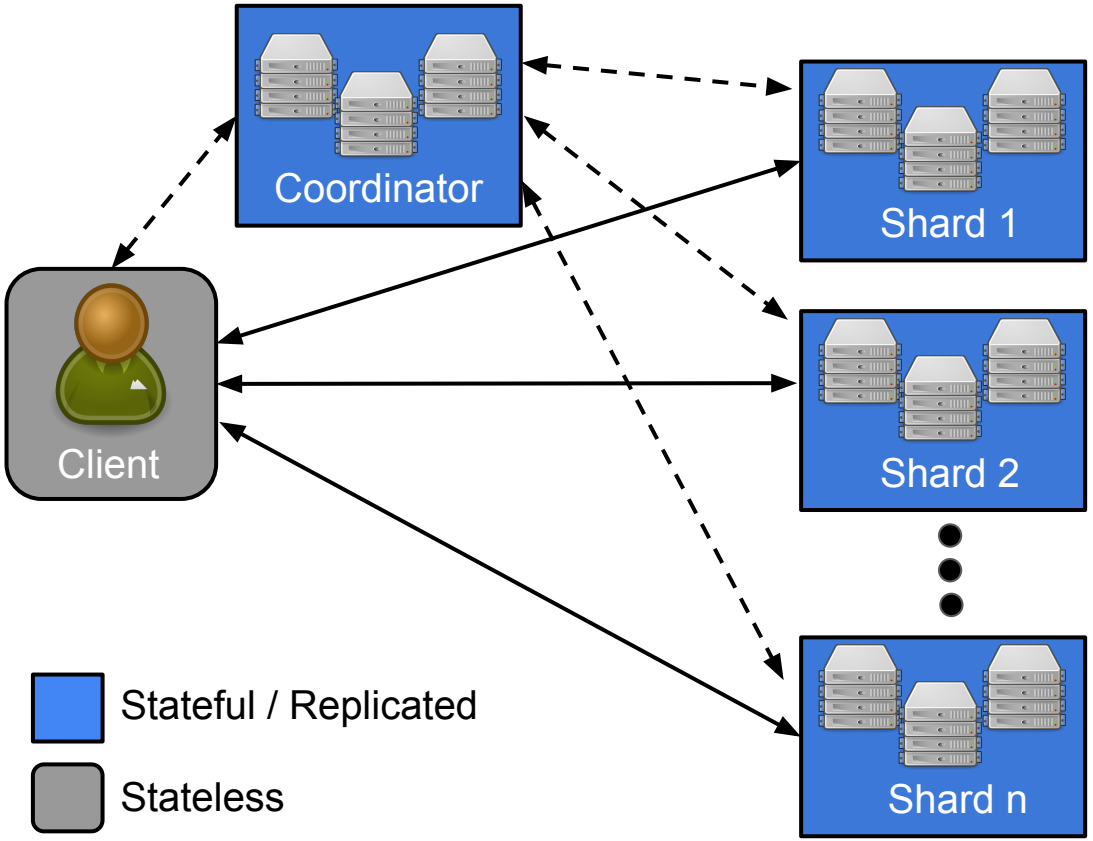


- **Fields** represent, structured or unstructured, data associated with an object
- **Functions** access or modify the state of an object
- Objects can only directly modify their own data

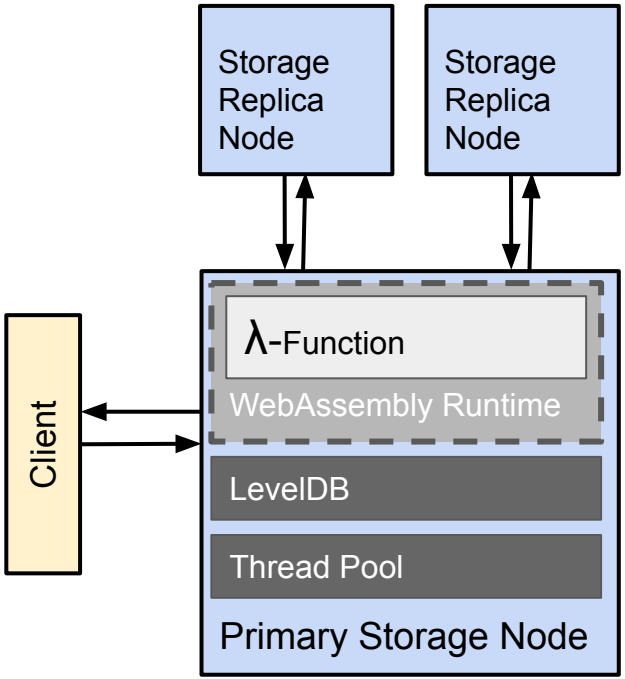
Example: Social network functionality as LambdaObjects

Datastore Design for Re-Aggregation

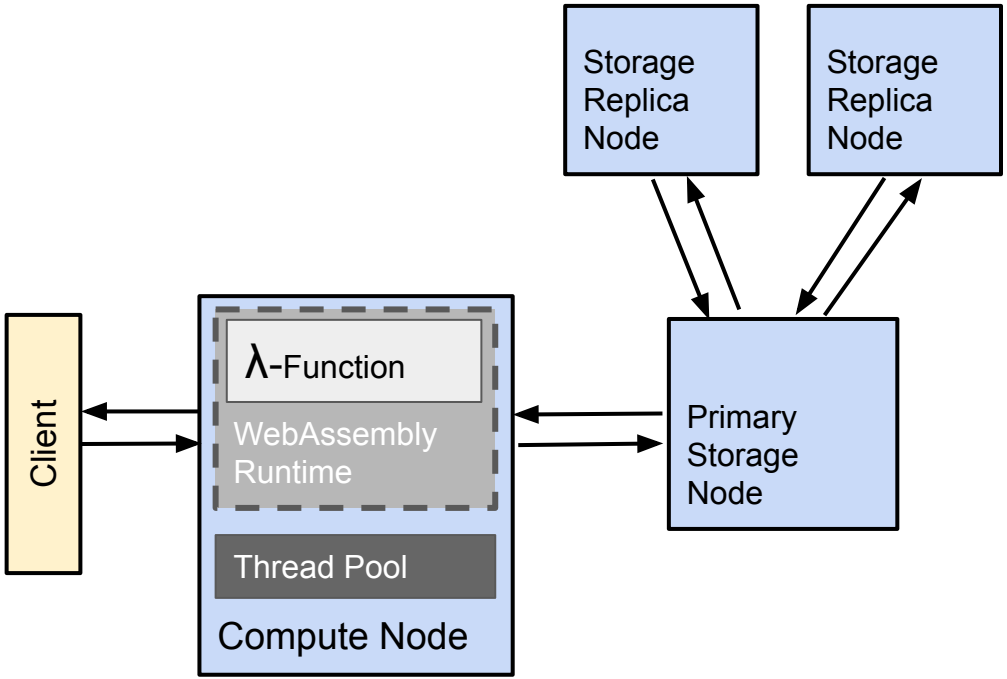
- Each LambdaObject is located at one shard
- Clients directly contact the shard to execute functions
- All nodes are connected to a **coordinating service** that detects and **manages failures**



Preliminary Evaluation Setup

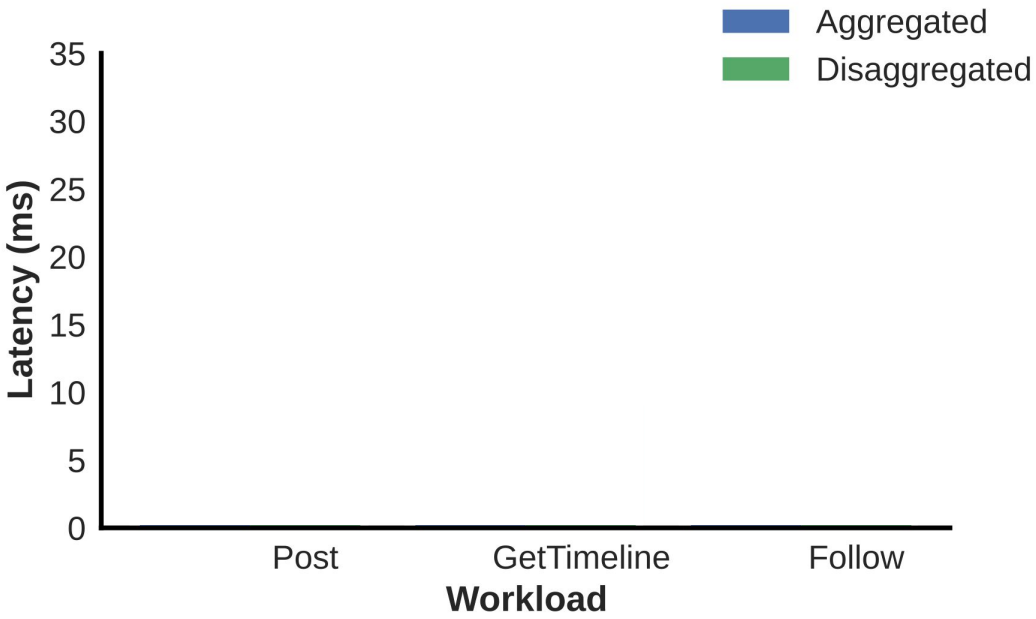


Re-Aggregated



Disaggregated

Preliminary Evaluation Results



Workloads:

- *Post:* Creates one post and updates all affected timelines (multiple function calls; read/write)
- *GetTimeline:* Retrieves the timeline for a specific account (single function call; read-only)
- *Follow:* Adds an account as a follower to another account (single function call; write-heavy)

Note: Latencies are generally low due to the use WebAssembly and lack of wide-area communication

Re-Aggregation: Challenges Revisited

- ✓ Determine which data and computation belongs together
- ✓ Design a compute-enabled datastore
- ✓ Support mutually distrusting functions
- ? Adapt to workload changes
- ? Provide transactional guarantees

Conclusion

Co-location of storage and execution enables serverless applications with **low latencies** and **strong consistency**

LambdaObjects are a **straightforward** and **efficient** abstraction for developers to build such co-located systems

Limitations:

- Not all use cases might fit this design
- Might not be beneficial for compute-heavy workloads
- Potentially expensive to port existing applications

 kaimast@cs.wisc.edu

 @cskama