Rendezvous: Where Serverless Functions Find Consistency

Mary Yang, Micaela Nomakchteinsky, Xiaoqing Zhou
Group 8

UNIVERSITY *of* WASHINGTON

Rendezvous: Where Serverless Functions Find Consistency [1]

- **Authors:**
- Mafalda Sofia Ferreira INESC-ID, Instituto Superior Técnico, Universidade de Lisboa João
  - Ph.D. student in Computer Science and Engineering at Insituto Superior Técnico, Universidade de Lisboa
- João Ferreira Loff INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
  - Ph.D. candidate at Insituto Superior Técnico, Universidade de Lisboa
- João Garcia INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
  - Assistant Professor at INESC-ID focused in Distributed Systems

- Institute of Systems and Computer Engineering - Research and Development
- Higher Technical Institute

Talk Outline

- Introduction
- Related Work
- Rendezvous Framework
- Key Contributions of Rendezvous
- Experimental Evaluation
- Conclusions
- Critique
- Gaps

UNIVERSITY *of* WASHINGTON

Problems and Challenges

- Functions used in Function-as-a-Service platforms are executed without state and independently of each other
- How can functions communicate with each-other, then?
  - Cloud providers have limited resources to mitigate this issue.
  - Developers have to rely on storage layers to share state between functions.
  - Developers want to have their data replicated across regions
    - Issues with delays in propagation of data
    - Access of outdated information
- **Question: Is there a way to maintain consistency of data in serverless applications using datastores, reducing the chance of reading outdated information?**

UNIVERSITY *of* WASHINGTON

Problems and Challenges

- **Geo-replicated data stores** – problem with latency and outdated information as discussed previously.
- **Lack of Coordination between functions** - Functions rely on indirect communication meaning there is not a way to track if data produced by one function is available to other functions that need it
- **Restriction of Datastore modifications** - cloud providers don't allow direct modification of the datastores, so developers can't insert and retrieve metadata. Lack of ways to monitor alterations to the objects in the database.
  - Ex: if application needs to add timestamp to data they might not be able to add this easily

UNIVERSITY of WASHINGTON

Background and Related Work

- **Communication in Serverless Applications**
  - FMI and Boxer – focused on quick and direct communication between functions, instead of relying on storage systems for certain tasks
  - Pocket – focused on sharing temporary info between functions
  - Glider - focused on enhancing data movement between functions, but not consistency [2]
- **Coordination in Serverless Applications**
  - Cloud providers offer orchestration techniques to build stateful workflows, however they might miss inconsistencies across different parts of a system.
- **Cross-Service Consistency**
  - Antipode: Works to ensure all parts of a distributed application are in sync. Keeps track of information about service dependencies and ordering of events.
  - FlightTracker - used by Facebook. Uses a central place to keep track of metadata to keep track of what all services are doing.
- **What is missing from previous work?**
  - The above techniques often lack a specific feature that Rendezvous combines to work with serverless systems
    - Consistency semantics, specialized coordination for serverless systems, efficient synchronization, and a centralized approach

UNIVERSITY of WASHINGTON

Generic Workflow Post-Notification Application

1. User Request for New Post:
User initiates a request to upload a new post, triggering a new writer function invocation.

2. Writing Post Data:
Writer function stores the post's data in write_p within the underlying datastore located in the EU region.
Data is asynchronously propagated to the US region.

5. Fetching Post Content:
The reader function retrieves the post's content from the US datastore (read_p) using the extracted identifier (p_id).

Notification event reaches the US before post data is available! Reader may retrieve outdated information!!!

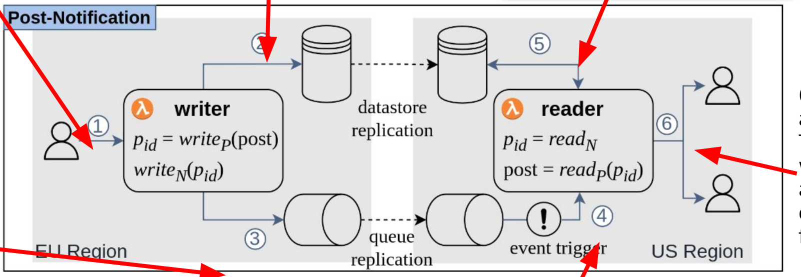


Fig 1. Example of a generic request workflow in the PostNotification application

3. Notification Event Queueing:
The identifier (p_id) generated from the write operation is included in a notification event.
Writer function queues this event in write_N.

6. Delivering Notification and Post Content:
The notification along with the post contents are delivered to the original poster's followers.

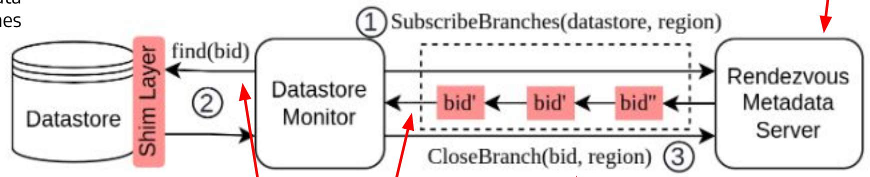
4. Triggering Reader Function:
Upon replication of data to the US region, an event triggers the reader function.
The post's identifier (p_id) is extracted from the event in read_N.

UNIVERSITY *of* WASHINGTON

Rendezvous Framework - Architecture

The **shim layer** extends typical datastore actions, adding Rendezvous metadata to track data branches

The **metadata server** stores branching details in requests, branches, and subscribers. It tracks open and closed branches per request and uses subscriber data structures to create queues of new branch IDs for the datastore monitor.



1 SubscribeBranches(datastore, region)
find(bid)
2
Datastore
Shim Layer
Datastore Monitor
bid'
bid'
bid''
Rendezvous Metadata Server
CloseBranch(bid, region) 3

Fig 2. Rendezvous Workflow

Rendezvous API for Closing Branches accompanies the DataStore Monitor for closing branches. Blocks read request through a Waiting call while write requests are not yet visible in the region of the request

Datastore Monitor monitors every new branch created with branch ID (bid) provided by the metadata server.
If the datastore is available for the region it notifies the Metadata server to close the branch

Registering new branches to a MetaData Server is done by calling a **Rendezvous API for Opening Branches**, specifying the targeted database

UNIVERSITY *of* WASHINGTON

Post-Notification Application with Rendezvous

A branch for the new write post is registered using the API for creating branches, then the metadata creates and opens the branch.

- OPENED status for both EU and US
- Returns bid

DataStore Monitor monitors the post and its bid

US is notified by the metadata server when a new branch is created

Writer function is enhanced to a call to the Shim layer which includes the branch ID (bid)

User Request for New Post: User initiates a request to upload a new post, triggering a new writer function invocation.

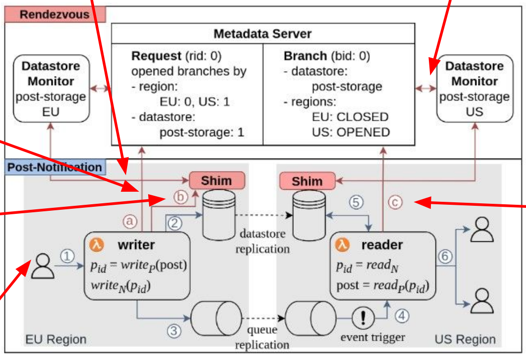


Fig 3. Integration with Rendezvous

A Wait Request function is called to block read operations while branches are OPENED in the US. When all branches are closed in the US , the WaitRequest is released, so the reader can read the post

UNIVERSITY *of* WASHINGTON

Integration, Optimizations, and Fault Tolerance

- **Integration with Cloud Platforms**
  - Cloud providers can allow developers to enable Rendezvous when configuring their function
  - Cloud providers would take care of wrapping read and write calls to datastores through the shim layer (write operations register new branches with included metadata, read operation automatically perform WaitRequests)
- **Optimizations**
  - Local access can be achieved by placing a metadata server node in each region the function is executed, reducing latency and fault tolerance
  - Developers can configure the WaitRequests through timeouts or bypasses.
- **Fault Tolerance**
  - Achieved through replication of metadata server across regions
  - Uses a call context to monitor connections to the database

UNIVERSITY *of* WASHINGTON

Key Contributions

- Registering the branch for the write post operation for all regions.
- The write operation to the datastore includes branch metadata (bid, regions, the datastore, and its OPENED or CLOSED status)
- Read operations are blocked with Wait Requests to ensure consistency across regions

Prevents inconsistencies in function outputs caused by weak consistency guarantees in geo-replicated settings. Rendezvous ensures data consistency and prevents the reading of outdated data.

Evaluation

Metrics:
1) consistency window
2) scalability

Experimental Setup:
- a pipeline of two Lambda functions
- the writer in EU and the reader in US
- notification queue: AWS SNS with notifications objects
- four backends: Redis, DynamoDB, MySQL, and S3
- deploy one server per region, provisioned in AWS EC2 t2.xlarge instances with 4 vCPU and 16 GiB RAM
- configured 1000 different posts for each evaluation run

Consistency window

The consistency window: the time between the writer function writing the post to the datastore in the EU and the post being read by the reader in the US

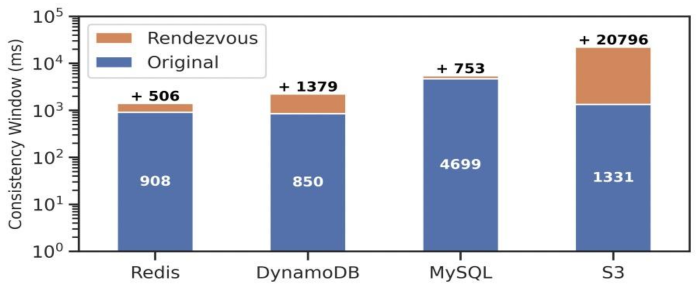


Fig 4. Variation of consistency window for each datastore with and without Rendezvous.

the baseline bars	the original setup
the above bars	the setup with Rendezvous.

- Varies a lot depending on the post's storage, which is a direct consequence of the duration of the WaitRequest call
- Most of the increase in the consistency window stems from the latency of replication specific to each datastore

UNIVERSITY *of* WASHINGTON

Scalability: Experiment 1 - Datastores

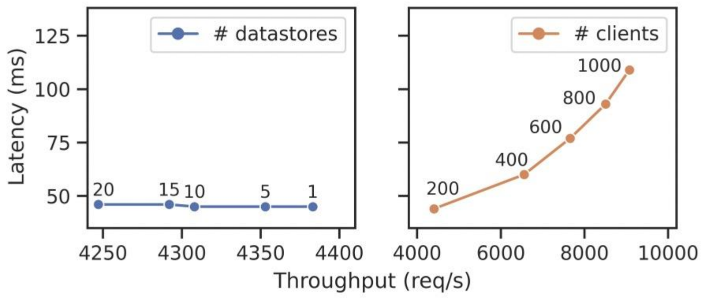


Fig 5. Relationship between throughput and latency observed in a Rendezvous metadata server by varying the number of (i) datastores, and (ii) clients.

Experimental Setup: 200 threads per client program, AWS EC2 t2.large, 2 vCPUs, 8 GiB RAM, in the same region as the server (EU)

Experiment 1:

- vary the number of datastores written to by a single function using a fixed number of 200 concurrent clients

Results:

- the throughput remains approximately the same when ranging from 1 to 20 datastores
- adequate performance if we do not expect functions to employ a number of datastores significantly larger than 20

UNIVERSITY *of* WASHINGTON

Scalability: Experiment 2 - Clients

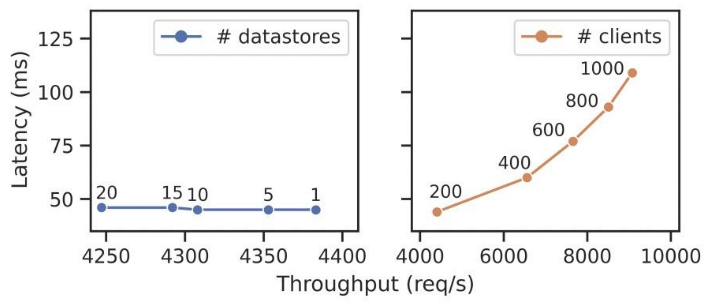


Fig 5. Relationship between throughput and latency observed in a Rendezvous metadata server by varying the number of (i) datastores, and (ii) clients.

Experiment 2:

- change the number of clients with a single datastore write operation

Results:

- the throughput increases with the number of concurrent clients, with a peak of approximately 110ms of latency with 1000 clients issuing close to 9000 requests per second.
- the number of clients has a greater impact on the latency than datastores

UNIVERSITY *of* WASHINGTON

Conclusions

- Addresses the issue of inconsistent executions due to the inherent latency of replication, when sharing state between functions.
- Enhances stateless functions by leveraging replicated datastores.
- Enables the synchronization of replicated data across different executions, aiming to eliminate inconsistent read operations while maintaining the consistency semantics of datastores.
- Can seamlessly integrate into cloud platforms and offer complete transparency to developers.

UNIVERSITY *of* WASHINGTON

Critique: Strengths

- Rendezvous framework provides consistency of data in FaaS applications without incurring much overhead compared to generic workflows.
  - Enforcing consistency has a low effect on overhead, never exceeding 70ms and averaging 7% across different datastores.
  - Can be smoothly integrated into cloud platforms, providing developers with full transparency.
- Concrete examples to illustrate the difficulties of maintain consistency across geo-replicated data stores
  - Used the post-notification workflow to introduce the data inconsistency problem.

    The paper also demonstrated the importance of the Rendezvous framework.
- Rendezvous: transparency & flexibility for developers
  - Enable more control and more synchronized state for users. Meanwhile, allow trading-off correctness for performance.

Critique: Weaknesses

- Rendezvous depends on **infrastructure components** such as the Datastore Monitor and the shim layer. It lacks fault-tolerance guarantees, and it's hard to recover from potential crash.
  - Possible solution: implement replication of these components.
- The authors used terms before explaining them:
  - "Consistency Window" is mentioned in sec 3.3, but the definition (explanation) is in sec 4.2.
- Poor figure layout
  - In the middle of a sentence, turn the page and insert the figure. Affect the reading experience.

Critique: Evaluation

- Limited datastore is tested.
    - Only 4 datastores were tested. S3 belongs to one of the cloud provider, AWS. But other datastores from other cloud provider could be tested. In this way, the application range of the Rendezvous framework is more wide.
- Insufficient test of scalability
    - Number of datastores is greater than 20 is not tested.

UNIVERSITY *of* WASHINGTON

Remaining Gaps

- Large customized workload.
    - For practical usage, now the application only supports AWS as cloud provider and 4 kinds of datastores. But there are many cloud providers and more datastore types. For every type, the datastore monitor needs to be implemented. And their updates may need more adjustments.
- Limited range of industry usage.
    - Usage of Rendezvous introduces more time overhead to ensure consistency. According to the evaluation part, this additional latency will influences with the number of clients. Thus the application scenarios are narrowed to high correctness requirements, tolerance to latency or small number of parallelled clients.

UNIVERSITY *of* WASHINGTON

Q&A

- Thanks for listening

References

1. Ferreira, M. S., Loff, J. F., & Garcia, J. (2023, October). Rendezvous: Where Serverless Functions Find Consistency. In *Proceedings of the 4th Workshop on Resource Disaggregation and Serverless* (pp. 51-57).
2. Barcelona-Pons, D., García-López, P., & Metzler, B. (2023, November). Glider: Serverless Ephemeral Stateful Near-Data Computation. In *Proceedings of the 24th International Middleware Conference on ZZZ* (pp. 247-260).