NSDI '23 Spring Paper #255 Reviews and Comments

===
Paper #255 Doing More with Less: Orchestrating Serverless Applications without an Orchestrator

Review #255A

===

Overall merit
----
2. Weak reject (I think it should be rejected, but I am fine if others want to accept.)

Reviewer expertise
----
3. Knowledgeable

Comments for author

This paper targets a problem in practice: current serverless workflow orchestrators are insufficient. The claimed insufficiencies of the current orchestrators include being centralized (thus lacking performance, fault-tolerance, and scalability), and having a high monetary cost. The paper proposes a new drop-in replacement called Unum. Unum works as a library working on the same machines as the serverless workloads; it leverages the provided storage (and its atomic semantics) to coordinate between nodes. Evaluation on AWS and Google Cloud shows an improvement in performance and a (huge) reduction in cost.

The paper is very easy to read, and gives the problem statement and the solution in a clear way. I enjoyed reading the paper. The strength and the weakness of the paper are both obvious to me. On the one hand, it targets a real-world problem and it achieves an excellent result. On the other hand, the solution is very simple and straightforward and lacks challenge, and the execution of the paper needs improvement. I would champion this paper if the execution was better (I don’t mind simplicity that much). The paper currently lacks/mis-states a few key things that are foundations of the claimed scientific contribution. I believe an improved execution will greatly strengthen this paper.

The first issue is that I find no evidence to back the important claim that current orchestration of cloud services is centralized. And it is hard to believe that a service with AWS scale has a centralized orchestrator. If this is real, please provide evidence. The intro cited [3,4,19,21] for this (“Cloud providers have deployed specialized centralized workflow orchestrators”). But I didn’t find descriptions of the orchestrator being centralized in these citations. I also searched online but didn’t find many descriptions on the architecture of AWS Step Functions. If I have to guess based on my experiences, I’d assume they implemented the service in a partitioned way, backed by a fault-tolerant storage such as DynamoDB, which is a common way of building such a service. Note that being external is different from being centralized, and perhaps what confuses you is that these services are external to the serverless worker nodes. If so, it means that a very important claim in the abstract and intro needs to be withdrawn, which is unfortunately a killer.

Despite the above misclaim on centralized services, I still believe the paper has a strong contribution that it provides a huge benefit on cost (the latency benefit is still good, but on its own might not be enough for an NSDI paper). The evaluation should provide more details on this. In particular, two questions need to be addressed.

First, what are the further cost details on the Step Functions? For example, what is the per state transition cost and how many state transitions are there? Figure 6 is amazing, it says that the major cost is actually in Step Functions, not the workloads themselves. It raises a question: should people use Step Functions at all, if they care about costs?

Second, you should compare the cost of other orchestration techniques, such as gg. This is important as it provides evidence, and a clear positioning of the scientific contribution of Unum: it is the first that can lower the cost of orchestration by up to an order of magnitude. If gg can also reduce cost, then you will have to trim the claim a bit, that Unum is the first to reduce cost of orchestration without deploying extra services. This unfortunately will undermine the contribution of Unum but has to be clear; being unclear hurts more.

Other questions on evaluation that are also important:

Why didn’t the evaluation show the cost on Google Cloud?

What is the cost of no orchestration (no information on this in Table 4)? Should people always use handcrafted workflow if they want the lowest cost?

What is the cost of using “driver function”? This seems to be the most non-intrusive way of implementing a flexible in-place orchestration. The main drawback claimed by the paper (S2) is that it has the “double billing” issue. This should be backed by the evaluation that it is an actual issue. But it is doubtful as in Figure 6, the User Code Duration’s cost is only a small part. The “driver function” I assume would belong to User Code Duration if implemented?

At last I want to discuss the design parts. These are actually the problems that I believe are easier to fix. The database providing atomic operations simplifies Unum’s design, but also undercutting the challenge Unum has to address and hence its novelty. The coordination goal of Unum, is less challenging than other works such as Beldi, which targets a transactional feature. However, there are still corner cases that need a few revisions, for example, the garbage collection. The current design says “in non-fan-out cases, once a node check-points its result, it can delete the previous checkpoint” (S3.5.1). I think there are corner cases that break the design. Say you have a chain A-B-C, once B has checkpointed its result, it will delete A’s result. However, consider there is a duplicated task of A, named A1, A1 finishes after A’s checkpoint is deleted. Then A1 will create a new checkpoint? There are two problems in this case. 1) how is A1’s new checkpoint garbage collected? 2) What if A1 is non-deterministic and it has a branch in its end, instead of launching B1, A1 launches a totally different node D1? All these cases need to be discussed and covered.

Another design choice that may need revision in discussion is the “exactly once” semantics. I think generally you need to clarify that the “exactly once” semantic cannot be guaranteed if there is an (external) side effect, because the design guarantees the semantic using checkpoints on results, not at launching/execution. I think you mentioned this very late, in related work?

In summary, I like the problem which the paper is targeting and the effectiveness in its solution. What I’d demand is a better description on its contribution, and a more detailed analysis on the cost advantage it has. I’d be happy to review it again in future submissions after these issues are fixed.

Review #255B

Overall merit

3. Weak accept (I think it should be accepted but I am fine if others want to reject.)

Reviewer expertise
----
2. Some familiarity

Paper summary

This paper proposes using a workflow orchestration library that is embedded into cloud serverless functions as an alternative to centralized orchestrators such as AWS Step Functions. This results in greater flexibility, i.e. the ability to implement application-specific optimizations, and cost savings.

Comments for author

As the serverless paradigm is gaining traction among application developers, the use of orchestrators that coordinate the execution of graphs of serverless functions is also increasing. Thus, your work is timely. Your introduction does a fair job of assessing the current situation with cloud-based orchestration services. Showing that such orchestration can be done within serverless functions themselves is an interesting contribution. I especially appreciate that you can run arbitrary Step Function workflows with the same guarantees. Though, I have trouble deciding whether this is mainly of academic interest or an approach that will advance the practical use of serverless workflows.

Here are some suggestions for what you could add to the paper to help convince me and the readers about the practicality and importance of this work:

Present a compelling, concrete example of an application that requires more flexibility than afforded by orchestration services. The intro mentions one example dealing with deterministic functions, but provides no details. Please explain the sort of application-specific optimizations that are both desirable and possible with your approach. As a later example, you state that a "fold" operator can be valuable for video encoding but is not supported in Step Functions. The paper does not explain why a Step Function user cannot construct a workflow that does something similar.

Explain why embedded orchestration can inherently scale better. You never define what you mean by "scalability" in the context of serverless workflows. And you do not address scalability in the evaluation section. So, it is not clear what limitations current orchestration services have regarding scalability.

Expand on your claims that you are able to run workflows with the same fault-tolerance. For example, a service-based orchestrator is able to re-start a workflow when machines (and the associated function invocations) fail, but it is not clear how your approach can offer the same type of recovery. Your section on fault-tolerance mostly deals with checkpointing and glosses over restarts.

Justify the importance of exactly-once execution. Your example of generating thumbnails from images in a photo library seems to contradict your insistence that workflow executors must guarantee exactly-once semantics. Generating a thumbnail more than once is not a problem, arguing that at-least-once semantics may be sufficient. Moreover, AWS Lambda does not even guarantee at-least-once execution, that is, it will perform a number of retries if a function fails but not indefinitely.

Demonstrate through your evaluation the fundamental benefits of decentralized (function-based) vs. centralized (server-based) orchestration. The presented results seem to show differences in the implementation of your approach compared to Step Functions but not whether these differences are inherent to either approach. For instance, you get benefits from increased parallelism in the execution of serverless functions. But why can't Step Functions achieve the same amount of parallelism? It would be great if you could explain the reasons for why Step Functions enforces limits since these same reasons might apply to your approach as well.

Sincerely,

Doug Terry

Review #255C
===

Overall merit
----
3. Weak accept (I think it should be accepted but I am fine if others want
   to reject.)

Reviewer expertise
----
2. Some familiarity

Paper summary

Unum is a decentralized orchestration service for serverless functions. Rather than relying on a fault-tolerance coordinator to manage a task execution graph, it maintains integrity of the graph execution through the atomicity guarantees available in cloud storage services like DynamoDB and Firestone. It also supports some operations, like folding, that are not currently available in centralized coordination services. Unum's evaluation compares its performance to AWS Step Functions, and considers its cost of execution on different cloud platforms.

Comments for author

Thank you for submitting your work to NSDI'23. There might not be a lot of ground left to cover with coordination services in general, but I do think there's a contribution to be made in decentralizing them, and I enjoyed the paper.

It seems like a key insight is that many operations that normally would require a centralized coordination service can instead be replaced with atomic operations at the storage layer. One follow up question I had about Unum is what does it assume about the storage layer. For example, would Amazon's S3, which has fairly weak consistency semantics, be sufficient to run Unum? S4 discusses some of these details, but it would have been nice to summarize this sooner in the paper. It seems like the short answer is that DynamoDB is required on the AWS platform. However, it was nice to see that Unum is general enough to support Google Cloud as well.

In S3.3, I understood how the checkpointing mechanism prevents corruption from duplicate executions, but how are failed functions themselves restarted? Is there some timeout mechanism? How is this implemented in a decentralized way? My basic assumption is this relies on Lambda's retry system, but it's not so simple to use:
https://docs.aws.amazon.com/lambda/latest/dg/invocation-retries.html

Overall, I thought that Unum had a sensible design. The evaluation was also decent, looking at both AWS and Google cloud, and comparing to Amazon Step Functions, a strong commercial baseline. However, I was disappointed that GCP was only considered for cost, and that performance results were not included. I also would have also liked to see how well Unum can handle failures in the evaluation, as this is the key area that contributes to complexity in Unum. Introducing some simulated failures, and comparing recovery to Step Functions, would improve the paper.

Detailed comments:

```
- In table 2, does map allow batching of multiple items? Is efficient to pass one at a
time to a function? Could FanIn be renamed to reduce?

- In S3.2, is there anything special or different about these ops versus a traditional
serverless orchestrator design? Do the ops help in anyway to support a decentralized
design?

- In S5.2.3, could Unum support preloading in the future? How would this work?

Nits:
- In S2, “bust” -> “burst”, “functio” -> “function”
- In S3, “fucnations” -> “functions”
- In S3.2, “less uncommon” -> “less common”



Review #255D
===========================================================================

Overall merit
-------------
3. Weak accept (I think it should be accepted but I am fine if others want
   to reject.)

Reviewer expertise
------------------
3. Knowledgeable

Paper summary
-------------
Unum provides a decentralized serverless workflow system. When provided a workflow
written in a high level language (e.g., AWS Step Function state-machine definition), Unum
compiles this into partitioned orchestration libraries, and relies on a serverless
consistent datastore (e.g., DynamoDB) to help the user functions execute the
orchestration logic in a decentralized way. By avoiding a centralized orchestrator, Unum
reduces the cost of deployment and improves runtime as well.

Comments for author
-------------------
Thank you for submitting to NSDI. I like the paper overall. It targets a timely practical
problem and it has good technical merit. I really liked the idea of leveraging serverless
consistent datastores already provided by FAAS providers for solving the
consensus/coordination needs of decentralized serverless workflow orchestration. With
improved presentation and with some refinements, this makes for an interesting
contribution to the conference.

Is this the first decentralized serverless workflow system? It seems so. Then the paper
should make a bolder claim for it. On the other hand, the paper should also do an honest
recount of why there has not been a pressing need for a decentralized workflow
orchestration solution. While there is a cost to running a central orchestrator, that
cost is often absorbed/compensated as the application using the serverless functions is
serving as the orchestrator. Moreover a centralized coordinator can allow dynamic
workflow execution, whereas Unum is constrained to static compilation of workflows.

There is more proof required before making a sweeping claim such as "Leveraging
decentralized orchestration, rather than centralized services can help performance,
resource usage, flexibility, and portability across cloud providers." Is there really a
networking or computation bottleneck with a centralized orchestrator (or the application
running in the cloud that is coordinating the serverless functions)? There is a
theoretical bottleneck, yes, but where is the practical bottleneck?
```

Related to the above comment, the evaluation of the paper can be improved further. While the evaluation provides some answers to the question of where the latency and cost improvements in Unum comes from, a finer granularity breakdown and a better presentation of the findings would help.

The related work discussion on Beldi and Boki should be expanded to provide more context, and discuss how their approaches compare to and complement this approach. The distributed shared log could serve as a good API to build a decentralized orchestrator as well, no?

Review #255E
===

Overall merit
----
1. Reject (I will argue to reject this paper)

Reviewer expertise
----
3. Knowledgeable

Paper summary

The paper proposes to decentralize the orchestrators of serverless workflows (which are currently centralized services). The argument is that the centralized orchestrators preclude users from making their own trade-offs between interactions of execution guarantees and performance, resource overhead, scalability and expressiveness. The authors present a decentralized serverless workflow orchestrator named Unum. Unum can translate existing workflow manifest into its IR which supports a few key primitives such as Invoke, Map, FanIn, Fanout, branches, etc. Unum relies on checkpoints to achieve the execution-only-once semantics where the checkpoints are stored in strongly consistent datastores. Unum can work with AWS Lambda and Google Cloud Functions. Evaluation shows that Unum can significantly improve the performance of certain types of serverless workflows and reduce the monetary cost.

Comments for author

Summary
+ Interesting alternative design of existing centralized serverless workflow orchestrator
- The motivation is unclear. I do not see how Unum helps users with tradeoffs between execution guarantees, performance, resource overhead, scalability, and expressiveness.
- The evaluation does not support the motivation either.
- Other orchestrator functionalities are overlooked, such as admission control and permission checks.
- Many important details are missing, e.g., the implementation of the frontend compiler, the representativeness of evaluated applications, etc.

Thanks for submitting the work to NSDI!

Unum is an interesting design for serverless workflow orchestrators. I'm not aware of a decentralized orchestrator, so the idea is novel. And I do believe that making the orchestrator decentralized can lead to benefits.

On the other hand, I find your motivation for Unum is weak and vague. The motivation statement is:

> centralized orchestrators also preclude users from making their own trade-offs between

available interactions or execution guarantees and performance, resource overhead, scalability and expressiveness.

However, throughout the paper, I didn’t find strong qualitative or quantitative evidence that supports this statement. You made two arguments:

> better for cloud providers as they need not develop and maintain yet another complex service

This does not make much sense to me. So, who is going to develop the decentralized orchestrator? The users? Or, some third parties? Eventually someone needs to “develop and maintain” the orchestrator, and cloud providers have the best interests to do it. Note that the decentralized orchestrator is more complex than the centralized one. So you are essentially increasing the complexity, rather than reducing it.

> It is better for developers as it gives applications more flexibility to use more performant, applications-specific orchestration optimizations and makes porting applications between different cloud platforms easier

I’m also not convinced by the more performant, optimization-friendly argument. I don’t see from the paper how the decentralized orchestrator can achieve it. In terms of portability, it is orthogonal to whether the design is centralized or decentralized. If your argument is that Unum provides an independent IR and that’s why it’s more portable. Then, the same thing can apply to centralized designs too.

Overall, you mentioned four things: (1) execution guarantee, (2) performance (3) resource overhead, (4) scalability, and (3) expressive.

(1) can be more effectively supported by a centralized design.

I don’t understand how (2) is improved from your experiments. In the evaluation, you mentioned that

> Unum performs comparably or significantly better than Step Functions in most cases owing to higher parallelism and a more expressive orchestration language,

What prevents a centralized orchestrator from achieving high parallelism? You mentioned that AWS Step Function limits parallel invocation of concurrent branches; why? I checked Ref [27] and it says,

> The default value is 0, which places no quota on parallelism and iterations are invoked as concurrently as possible.

Intuitively, Unum needs to pay the overhead of checkpointing, which is not needed by a centralized design. How does it affect the performance?

For (3) expressiveness, the ExCamera example in which you optimize the data dependency is interesting. OTOH, I don’t find it supports your argument too strongly, because it can be achieved by a centralized orchestrator also. The key point you are making is that the existing orchestrator is hard to customize, which is orthogonal to whether the design is centralized or decentralized. Is it fair to say that the expressiveness can be provided by a centralized design also (by providing more fine-grained control)?

Besides the ExCamera example, what are the other limitations of expressiveness of existing orchestrators?

For (4) scalability, what are you referring to? It's not evaluated or discussed.

I’d like to point out that orchestrators typically perform many other functionalities, such as admission control to prevent overload and permission checks. How do Unum achieve

those? One question I have is how to prevent malicious or unauthorized users from
invoking protected functions? How do you achieve security?

I find that there are many important details missing in the current draft.
How is the frontend compiler implemented? Given that your IR is likely more expressive
than existing offering; how do you translate existing manifest into your IR?
What are the evaluation applications? Are those representative serverless workflows? Who
implemented them? How big are they (e.g., how many functions and what is the length)?
What are the workloads you use?
What exactly is your IR? What you described is less a complete instruction set but more a
few APIs. Why do you think your IR is expressive enough?

Comment @A1 by Reviewer C

This paper was discussed in detail during the PC meeting. The reviewers appreciated the
paper's interesting approach of relying on storage and atomic operations to support a
fault tolerant orchestration service. However, they were concerned about the lack of
evidence that the approach Unum takes has advantages over a more traditional, logically
centralized orchestration service. Ultimately, the PC decided to accept the paper, but
they expect the authors to work closely with the shepherd to resolve the following
issues:

- Scale back (or remove if suggested by the shepherd) any claims about the benefits of
decentralization that are not strictly supported by evidence in the paper
- Clarify that existing orchestrators are not strictly centralized either (e.g., they can
rely on sharding, replicated state machines, etc.), and emphasize actual differences in
Unum's design.
- Clarify how other key functions like access control, rate limiting, and restarting
failed work can be handled in Unum.
- Discuss gg as an alternative option in relation to your claims about cost.