Serverless Multi-Query Motion Planning for Fog Robotics

Raghav Anand, Jeffrey Ichnowski, Chenggang Wu, Joseph M. Hellerstein, Joseph E. Gonzalez, Ken Goldberg

***Abstract*— Robots in semi-structured environments such as homes and warehouses sporadically require computation of high-dimensional motion plans. Cloud and fog-based parallelization of motion planning can speed up planning. This can be further made efficient by the use of "serverless" on-demand computing as opposed to always-on high end computers. This paper explores parallelizing the computation of a sampling-based multi-query motion planner based on asymptotically-optimal Probabilistic Road Maps (PRM*) using the simultaneous execution of 100s of cloud-based serverless functions. We propose an algorithm to overcome the communication and bandwidth limitations of serverless computing and use different work-sharing techniques to further optimize the cost and run time. Additionally, we provide proofs of probabilistic completeness and asymptotic optimality. In experiments on synthetic benchmarks and on a physical Fetch robot performing a sequence of decluttering motions, we observe up to a 50x speedup relative to a 4 core edge computer with only a marginally higher cost.**

I. Introduction

Many robotics applications, from home automation to warehouse order fulfillment can benefit from access to a fast motion planner that allows the robot to interact in a physical space. For robots in cluttered environments that have many degrees of freedom, planning can be computationally challenging [1]. Moreover, time to plan a motion can widely vary depending on the complexity of the robot and environment. An always-on high-end computer, whether on-premises or in the cloud, can be a sub-optimal use of resources [2]. Consider a robot tasked with cleaning desks in an office space (see Fig. 1). Due to the variability of obstacles the robot has to replan motions for each desk. When decluttering a single desk, it should reuse computations rather than replanning every pick and place. Mobile manipulators like the Fetch often need to recompute motion plans in the same environment due to variations in the positioning of the frame of the manipulator-arm based on inaccurate driving wheels or obstacles blocking navigation. In this example, computing demands vary dramatically between the navigation to a desk and the decluttering of a desk. Navigation is relatively inexpensive as finding paths in a 3-dimensional space is not difficult. Whereas planning manipulator arm motions to grasp objects requires solving 6-dimensional or higher problems that need exponentially more computation. Similarly robots in a warehouse have to plan manipulator arm trajectories for varying motions within a single work cell.

We propose applying serverless computing to robot motion planning problem. Serverless computing allows developers to register functions in the cloud without managing the infrastructure associated with executing these functions [3]. As serverless computing is focused on single-function execution, it is often called Function-as-a-Service (FaaS). Functions run on-demand on various cloud, fog, and edge systems and are billed in small time units (e.g., 1 ms [4]). Computational workloads in motion planning for home, office, and warehouse scenarios are often intermittent due to time spent in low-dimensional navigation problems, performing reaching motions, or being offline. Although Infrastructure-as-a-service (IaaS) gives developers access to large amounts of compute, elasticity is at the virtual machine (VM) level. This can be difficult to manage and thus result in over or under-provisioning of resources [5]. As such, the computational requirements for motion planning match well with the serverless paradigm of elastically scaling compute resources to minimize cost and meet demand [6]. Although each serverless unit is limited in its computational capabilities, we propose an efficient parallel algorithm that can take advantage of the simultaneous availability of 100s of compute units to achieve large speedups. It is common to refer to a single execution of a serverless function as a *lambda* [4].

Serverless computing does have limitations, including an inability to directly communicate with other computers, no

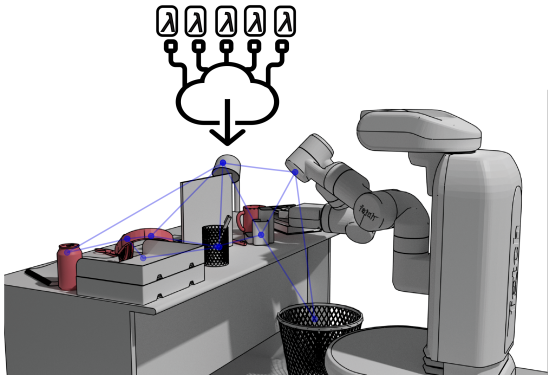

Fig. 1. A Fetch mobile manipulator robot organizes and declutters a desk with a sequence of motions computed through on-demand parallel computation with serverless fog robotics. In this scenario, after the robot approaches the desk, it picks and places the objects shown in red. Every decluttering sequence, even for the same desk, requires a new motion plan computation due to changes in the obstacle environment, either due to changes of objects, or due to inaccuracy or blocked approaches to the desk. To quickly start the sequence of tasks, the robot computes a single graph **(shown in blue)** of obstacle-free motions using 100s motion-planning serverless functions (aka *lambdas* λ) for a short duration. Once computed, the robot follows motions between points on the graph. Since serverless computing is billed in 100 ms units, the proposed approach costs the same whether using 1 serverless computer for 500 seconds, or 500 serverless computers for 1 second.

University of California, Berkeley. {raghav98, jeffi, cgwu, hellerstein, jegonzal, goldberg}@berkeley.edu

direct permanent storage between executions, and unpredictable delays in execution [7], [8]. We overcome some of these limitations but note that future serverless offerings may relax these restrictions [9], [10], [11]. The proposed method may scale better and become more cost-effective in the future with no change to the algorithm. Additionally, ongoing work on serverless Edge computing [12] brings compute closer to robots and can allow for seamless scalability with minimal network overhead.

This paper proposes a method that leverages the elasticity of serverless computing to parallelize computations of complex multi-query motion plans on-demand. We show that the robot can flexibly allocate lambdas to each problem allowing it to allocate *more* parallelism to compute motion plans faster with a marginal increase in cost, . The proposed method scales well up to 128 concurrent lambdas which suggests the cost-effectiveness of this method. The main limit to scaling beyond 128 lambdas is startup delays of functions. To test the potential for further scaling we control for these delays and scale up to 512 concurrent lambdas. We observe up to a 52x speedup compared to a 4-core local baseline for a physical Fetch robot [13] decluttering scenario and up to a 100x speedup on synthetic motion planning benchmarks.

This paper makes the following contributions:

1) a distributed parallel algorithm for computing Probabilistic Road Maps "Star" (PRM*), probabilistically-complete and asymptotically-optimal motion planner, using serverless computing
2) an implemented version of the algorithm on Amazon Web Services FaaS "Lambda" environment
3) a method for time and cost bounded allocation of resources for motion-planning for generating graphs of a given size
4) experiments in simulation and on a physical Fetch mobile manipulator robot that suggest that the proposed algorithm provides significant speedups against local baselines.

II. Related Work

In this section, we provide background on sampling-based motion planners and prior work on parallelizing them. We also describe serverless computing and fog robotics.

A. Sampling Based Motion Planners

Sampling-based motion planners solve motion-planning [14] problems by generating random robot configurations and connecting them into a graph of feasible motions. Planners such as PRM [15] and RRT [16] are probabilistically complete, meaning that with enough time, they will find a solution with probability 1. With attention to sampling and connection strategy, these planners can be asymptotically-optimal (e.g., PRM* and RRT* [17] and SST [18]), meaning that with enough time, they will find an optimal solution with probability 1. In some scenarios, finding a single solution to a motion planning problem or *single-query* is sufficient, while in other scenarios it can be beneficial to precompute a *multi-query* graph or road map of motions that can later be quickly searched with different start and goal configurations.

Amato et al. [19] showed that sampling-based motion planners are well-suited for parallel computation. Prior work on parallelizing these motion planners explored building a single graph in shared memory with locks [20] and without locks [21], in distributed memory [22], [23], [24], resource-aware motion planning [25] and more.

B. Serverless Computing and Fog Robotics

Serverless computing has gained wide attention in recent years in both academia and industry for a variety of workloads [26], [27], [28]. Compared to server-based computing, where users provision and compute with virtual machines (VM), serverless computing has two key advantages. First, it abstracts away the notion of servers; users register functions and invoke them. This simplifies the deployment process as users do not need to manually provision VMs. Second, serverless platforms automatically adapt to workload changes and users only pay for the compute allocated during the function execution [3]. In our setting, a robot's computing requirements sporadically spike, making serverless computing an attractive option.

Cloud-based computation for robotics shows promise in offloading compute-intensive processes from a robot's on-board computer [29] to the public cloud or servers on the Edge, allowing robots to have low-power CPUs and light-weight batteries to power them. The computational requirements of high-dimensional motion planning [1] make it a good candidate for distributed cloud-based computation [30] - for example, Plaku et al. [31] present a planner that solves problems that exhaust resources on a single machine. In prior work, Ichnowski et al. [2] showed that serverless computing of tree-based single-query motion planners can dramatically speed up motion planning. However, tree-based planners need to replan for every new problem, even if the robot is operating in the same environment. This paper leverages a graph-based planner that enables reusing exploration from previous motion plans.

III. Problem Statement

In this paper, we parallelize the computation of multi-query motion planning using cloud-based serverless computing.

A. Multi-query Motion Planning Problem

Let $\mathbf{q} \in \mathcal{C}$ be the complete specification of a robot's degrees of freedom (e.g., joint angles, or position and orientation in space), where $\mathcal{C}$ is the set of all possible configurations. Let $\mathcal{C}_{\text{obstacle}} \subset \mathcal{C}$ be the configurations that are in collision or disallowed, and the remaining configurations $\mathcal{C}_{\text{free}} = \mathcal{C} \setminus \mathcal{C}_{\text{obstacle}}$ is the free space. Given a start configuration $\mathbf{q}_{\text{start}} \in \mathcal{C}_{\text{free}}$ and a goal configuration $\mathbf{q}_{\text{goal}}$, the objective of motion planning is to find a sequence $\tau = (\mathbf{q}_0, \mathbf{q}_1, \ldots, \mathbf{q}_n)$ such that $\mathbf{q}_0 = \mathbf{q}_{\text{start}}$, $\mathbf{q}_n = \mathbf{q}_{\text{goal}}$, and paths between all consecutive pairs of points in τ are collision free.

The objective of multi-query motion planning is to precompute a data structure that allows for the efficient computation of τ given changing $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$.

Given a cost function $d : \mathcal{C} \times \mathcal{C} \to \mathbb{R}^+$, let $c(\tau) = \sum_{i=0}^{n-1} d(\mathbf{q}_i, \mathbf{q}_{i+1})$. The objective of optimal motion planning is to compute a τ that minimizes $c(\tau)$.

B. Serverless Computing Environment

The robot has an onboard computer and networked access to a cloud-based serverless computing service. The serverless computing service allows for an unbounded number of concurrent executions of single functions, with the limitation that they cannot store state between executions, cannot accept inbound network connections and have bounded runtime. The goal of serverless multi-query motion planning is to perform the parallel precomputation step of multi-query motion planning, allowing for faster computation at the expense of more parallelism.

IV. Method

We start with background on PRM and PRM* algorithms, followed by a discussion of the challenges that arise from parallelizing PRM* using serverless computing. We then propose a parallel serverless sampling-based motion planner.

A. Probabilistic Road Maps (PRM) and PRM* Background

The Probabilistic Road Maps (PRM) [15] motion planner randomly samples configurations to build a connectivity graph of the environment. This graph is searched to find paths between any two points. PRM samples n configurations in $\mathcal{C}_{\text{free}}$ and attempts to connect k_{prm} pairs of configurations provided there is a collision-free path between them using a local planner. In the query phase, a shortest-path search (e.g., Djikstra's) computes a path connecting start and goal configurations. PRM is probabilistically complete. Karaman et al. [17] propose PRM*, with $k_{\text{prm}} \geq k^*_{\text{prm}}$, where $k^*_{\text{prm}} = e(1 + \frac{1}{d}) \log n$, and d is the dimension of the planning problem, PRM* is asymptotically-optimal.

B. Parallelizing PRM* using Serverless Compute

Parallelizing PRM* over a serverless environment presents additional challenges compared to local methods of parallelization [14] due to network overhead that makes information sharing prohibitively expensive. Thus, sharing nearest neighbor data structures [32] between lambdas is infeasible.

To work around the limits of serverless computing, specifically statelessness and only allowing outbound network connections, a coordinator algorithm is defined. This coordinator runs on a separate computer that allows inbound connections and can keep state. If the robot has a public IP address, it can run the coordinator algorithm and bypass the provisioned server. However, the coordinator algorithm has low CPU and memory requirements, so it can be a lightweight cloud instance with a far lower cost than the compute-intensive resources required for motion planning.

Algorithm 1 Coordinator Algorithm: A packet in the work queue is a group of vertices to be connected to the graph

```
G = (V = ∅, E = ∅)
work_queue = initWorkQueue(packet_size)
for lambda_id in num_lambdas do
    work = work_queue.pop()
    initializeLambda(lambda_id, work, seed)
while not done do
    for lambda in lambdas do
        Receive new edges E_l from lambda
        E = E ⋃ E_l
        Send next work packet to lambda
    if work_queue empty then
        Send done to all lambdas and return graph to robot
```

Algorithm 2 Lambda Algorithm

```
i = 0
(start_index, end_index) = work
nn = nearest neighbor structure
rng = random_state_generator(seed)
while not done do
    while i < end_index do
        q_rand ← rng.sample()
        if q_rand ∈ C_free then
            i = i + 1
            if i > start_index then
                Update k_prm
                for q_near in nn.near(q_rand, k_prm) do
                    E_l = connect(q_rand, q_near)
        nn.add(q_rand)
        Send E_l to coordinator
    work ← poll coordinator
```

C. Serverless Algorithm

To parallelize the computation of the PRM and minimize communication costs, Alg. 1 exploits the determinism of pseudo-random number generators to create a deterministic sequence of points when provided with a particular seed. As long as all lambdas are initialized with the same random seed, they will sample the same set of points. The sampling stage of the PRM* algorithm is orders of magnitude faster than the nearest neighbor queries and the connection of edges. For instance, sampling and validating 1000 points for an 8-dimensional space took 0.063 seconds, whereas connecting the edges for the above samples took 6.194 seconds. The key trade-off in this algorithm is to perform repeated fast sampling instead of slow communication of samples. As the size of the graph increases and the amount of serial sampling work decreases as a proportion of the total work, Amdahl's law [33] tells us that the theoretical maximum parallel efficiency goes up.

In Alg. 2 all the lambdas perform the sampling step for the required graph size. Vertices get unique IDs using a counter. These IDs are shared between the coordinator and lambdas implicitly through the common seed in the sampling process.

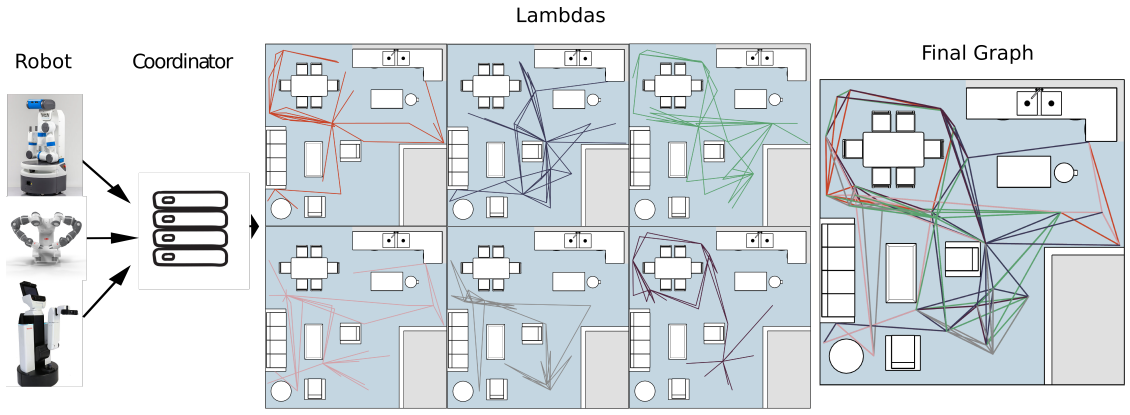

Fig. 2. A central coordinator handles initializing lambdas and maintains open connections to them to allow communication between lambdas. Note that the coordinator need not be a very large instance as it performs a network bound task. Multiple robots can also reuse a single coordinator for maximum efficiency. Each of the 6 lambdas connects a subset of the edges in the sampled vertices. These edges are sent to the coordinator which combines them into the complete graph on the right.

This allows lambdas to send edges to the coordinator using vertex IDs instead of d-dimensional state, resulting in lower bandwidth communication.

Vertex IDs are divided into work packets using different work-sharing methods (Sec. IV-D). A work packet has start and end vertex IDs that indicate the vertices to be connected to the rest of the graph. Each lambda in Alg. 2 is initialized with a work packet and begins vertex sampling and edge connection. Any new edges are sent to the coordinator (Alg. 1) which responds with another work packet or a termination message. Fig. 2 shows the end-to-end behavior of the algorithm and the contribution of each lambda to the graph.

Alg. 2 maintains probabilistic completeness and asymptotic optimality of PRM*. The common random seed ensures that all lambdas generate the same sequence of vertices $(v_1, \dots v_n)$. Similarly, the sequence of edges $(e_1, \dots e_n)$ is identical to the serial version of PRM* as each lambda generates the same edge-set for its vertices. Since the same vertices and edges are present in the graph as in the serial algorithm, Alg. 1 inherits probabilistic completeness and asymptotic optimality from PRM*.

Additionally, this algorithm maintains the computational complexity of the serial PRM* algorithm. Karaman et al. [17] showed that the complexity of PRM* is $\mathcal{O}(n \log n \cdot \log^d m)$, where m is the number of obstacles in the region. The additional work Alg. 1 performs is the repeated sampling done by each lambda. In the worst case, each lambda samples every vertex. Given l lambdas, this results in an additional $l \cdot n$ amount of work being done and the resultant complexity is $\mathcal{O}(n \log n \cdot \log^d m + l \cdot n)$. However, the number of lambdas is much smaller than the number of samples ($l \ll n$), and the last term drops out of the complexity analysis, giving back the original complexity of the PRM* algorithm.

D. Work Sharing

We define 5 work sharing methods:

No Work Sharing Each lambda uses a packet size of $\frac{\text{num\_vertices}}{\text{num\_lambdas}}$. This requires no communication with the coordinator and lambdas will terminate after processing the first work packet and sending edges to the coordinator.

Cyclic Work Sharing Each lambda processes vertices that have the property that $\text{vertex\_id} \mod \text{num\_lambdas} = \text{lambda\_id}$. This requires all lambdas to sample nearly all vertices, but distributes work more evenly.

Synchronous Work Sharing After sending edges to the coordinator the lambda blocks communication until a new work packet is received. This can result in idle time between two work packets but ensures that work is distributed evenly.

Asynchronous Work Sharing Lambdas poll the coordinator for a work packet shortly before processing the current work packet. This reduces network overhead and idle time as packets are available in the network queue of lambdas, but results in less even distribution of work than the synchronous method.

Equal Work Amount Per Packet The cost to add a new vertex to the PRM graph goes up with the number of vertices since k^*_{prm} grows. Small packet sizes in previously discussed work-sharing methods make the distribution of work between lambdas more even, however, this leads to additional communication costs. Packets that have an equal amount of work instead of an equal vertex count can result in good work distribution without additional communication.

In order to find packets with equal work (rather than equal vertex counts of previous methods) we estimated the work per new vertex. The time spent connecting edges for each vertex is proportional to the number of edges that have to be checked for collisions, which is determined by k^*_{prm}. Since k^*_{prm} is proportional to $\log n$, the work to connect each vertex should grow as $\log n$. To test this hypothesis, the connection time was measured (in Fig. 3), and the resulting graph roughly matched a $\log$-distribution.

This $\log$-like work distribution is leveraged by creating variable size packets with the property that the $\log$-sum of the vertex IDs in them are equal. Under the assumption that the relative time to connect each vertex is approximately proportional to the $\log$ of its vertex ID, this creates packets that

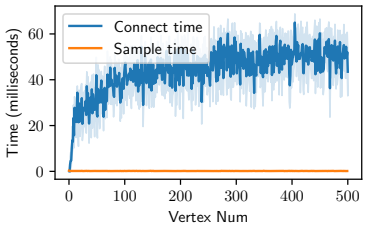

Fig. 3. The blue line refers to the edge connection time for the vertex and the orange line refers to the sampling time for the same vertex. The sampling time is much lower than the edge connection time. The edge connection time roughly follows a log function

have the same approximate amount of work. This method does not require any communication with the coordinator.

V. Experiments and Results

We experiment with the proposed system on SE(3) synthetic benchmarks from OMPL [34] and physical Fetch decluttering tasks, running on the Amazon AWS Lambda serverless computing environment. The coordinating server runs on a c5.xlarge instance (two 64-bit Intel Xeon cores) in the same region as the lambda processes. All experiments are run for 11 trials while varying the random seed.

A. Work sharing comparison

To compare the different work-sharing methods discussed, we generate a graph of 17000 vertices and approximately 700000 edges with each work-sharing method while varying the number of lambdas and the number of packets sent. We compare all methods across two metrics: the total cost of the serverless execution and the termination time of the algorithm. The first row of graphs in Fig. 4 shows the results for all scenarios.

Comparing asynchronous and synchronous methods across different scenarios, we observe from Fig. 4 that the former outperforms the latter in both cost and end time for nearly all packet sizes due to the lower communication overhead. This supports the hypothesis that communication costs exceed repeated sampling costs in a serverless environment.

Comparing the performance of different packet sizes within the asynchronous method, we observe that a moderate packet size performs the best in terms of end-time—too small of a packet size results in high communication costs on the coordinator, while too large of a packet size results in uneven work distribution that causes some lambdas to straggle. However, the cost of execution monotonically decreases with increasing packet size. This is because the slowest-to-complete lambda determines the end time of the algorithm, while the aggregate execution time determines the cost. A large packet size results in fewer packets that allow a greater proportion of lambdas to finish early due to the low work allocated to them which brings down the overall cost.

No-work-sharing outperforms asynchronous and synchronous methods on cost for various packet sizes, however, the cheapest asynchronous method is usually cheaper to run. This is because no work sharing can be reframed as synchronous work sharing with one large packet, and large packet sizes result in reduced costs. However, no-work-sharing suffers from a worse end-time than asynchronous methods due to poor work distribution. Cyclic work-sharing methods perform worse on both metrics than other methods because cyclic work-sharing requires every lambda to sample nearly all vertices in the graph.

Finally, log-based work-sharing (using packets with equal work) on the Fetch scenarios for high numbers of lambdas (128 or above) finishes as quickly as the asynchronous work-sharing method and has nearly the same cost as no-work-sharing with fixed packet sizes: the absence of communication lowers the cost while the approximately equal work in each packet allows for the quicker end times. Additionally, the cost of log-based work sharing is only slightly greater than no-work-sharing. However, at lower numbers of lambdas and for certain scenarios (like the SE(3) simulations), log-based work sharing is outperformed by asynchronous work-sharing methods. This is likely due to large deviations from the predicted log-growth of work for SE(3) scenarios that cause an uneven work distribution.

B. Scaling with Lambdas

We then experiment to measure the speedup provided by more lambdas. Ideal scaling means that a doubling of lambdas leads to a halving of runtime. However, due to startup and network overhead, and repeated sampling work, real-world scaling incurs performance penalties.

Fig. 4 compares 4- and 8-core local baselines (running PRM* on the robot's CPU) against Alg. 1 running on 16 to 128 lambdas. Scaling beyond 128 lambdas is difficult with the current serverless offering due to startup overhead—simultaneously starting up 256 lambdas takes 5s, which is nearly the runtime of the algorithm on 128 lambdas. Since the startup time is itself greater than the algorithm runtime, Fig 4 omits showing results for more than 128 lambdas. We hypothesize that this startup delay will be eliminated in the future [9], [10], and also simulate lambdas starting without the delay. We show results for 128*, 256*, and 512* lambdas that simulate no delayed startup with an asterisk.

To quantify the speedup of the algorithm, we measure the parallel efficiency of k lambdas which can be defined as $\frac{\text{time for m lambdas}}{\text{time for k lambdas}} \cdot \frac{m}{k}$ where m refers to the number of lambdas being compared against. A parallel efficiency of 1 indicates ideal scaling.

Fig. 4 shows that using 128 lambdas has a mean parallel efficiency of 0.77 as compared to 4 cores. As the number of lambdas increases, we observe that the proportion of time spent in sampling and startup increases and causes the efficiency to drop. When controlling for startup overhead, we observe that the algorithm continues to scale to 512 lambdas. 512 lambdas on the Fetch scenarios has an end-time of only 2.9s compared to the 157.2s end time of a 4-core local baseline, a 52x speedup. SE(3) scenarios scale even better with 512 lambdas finishing in 1.43s compared to the 153.5s of the 4-core local baseline, a 106x speedup.

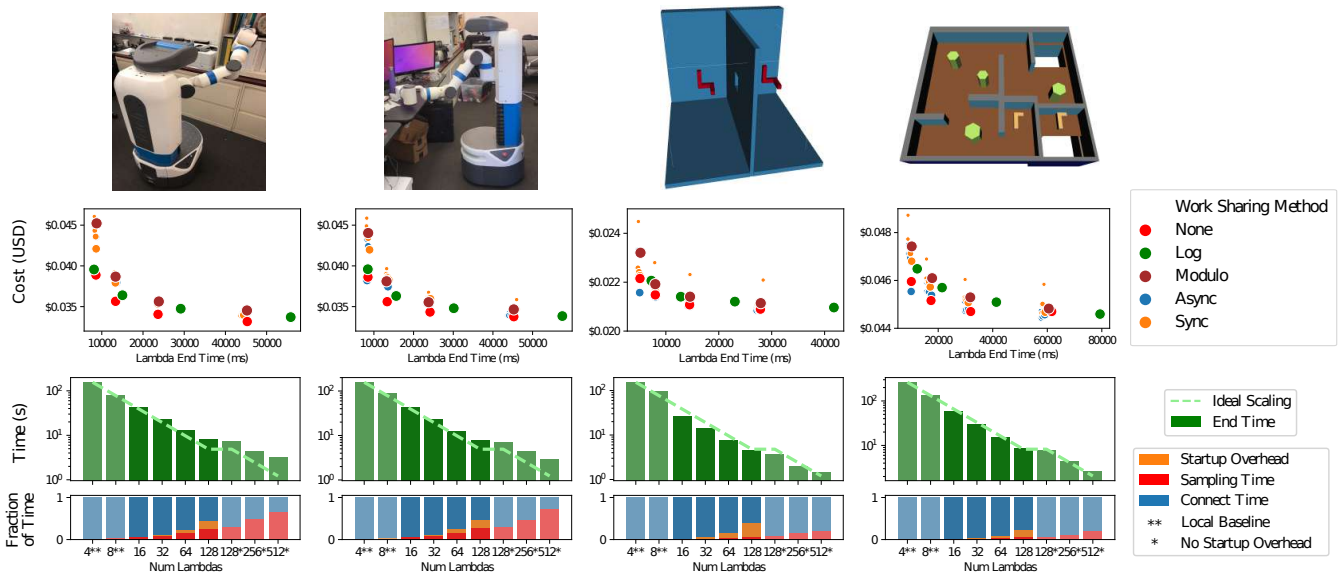

Fig. 4. Experiments are run on real-world decluttering scenarios (with the Fetch robot) and synthetic benchmarks. The first row for each scenario depicts the tradeoff between cost and end-time that a user can make: the size of the dot indicates the number of packets sent. Higher numbers of lambdas always finish quicker but have a marginally higher cost; and the best work-sharing method is scenario dependent. The second row for each scenario shows the time scaling of the algorithm with an increasing number of lambdas: as the number of lambdas increases the startup overhead and sampling time costs dominate the overall computation that causes a reduction in parallel efficiency.

C. Choosing Optimal Parameters

In a real-world scenario, people are interested in tradeoffs between cost and end time. We plot the cost of execution in USD against the time across all lambda counts in Fig. 4. The ideal position for a point on the graph corresponding to the lowest end time (left) and lowest cost (down) is in the bottom-left. A point is on the bottom-left of any neighbors is a strictly better choice.

One can traverse the graphs in Fig. 4 to pick optimal parameters for a specific application. For the Fetch scenario, the log-based work-sharing method is to the bottom-left of the asynchronous and synchronous work-sharing methods due to its low cost. However, in these scenarios, no-work-sharing is marginally cheaper than the log-based method with the penalty of a higher end-time. Thus depending on the unit economics of the application, the choice can be made between log-based work-sharing and no-work-sharing. In this example no-work-sharing is suitable where the unit economics don't allow a marginally higher cost for quicker end times, otherwise, log-based work sharing is preferable. Similarly lower lambda counts can result in a lower unit cost, but at the penalty of much worse end times.

Another perspective involves viewing idle time on the robot as a lost opportunity cost. If the opportunity cost can be quantified, then the time saved by using a more expensive work-sharing method or more lambdas can translate to cost savings. For example, in the Fetch scenario, the log-based method is faster than no-work-sharing by 0.2s but costs $0.001 more. If 3 minutes of robot idle time is worth more than $1, then the log-based work-sharing method is superior due to the additional work that can be performed by the robot.

VI. Conclusion

We propose using cloud-based serverless computing to rapidly compute a probabilistically-complete and asymptotically-optimal road map for multi-query motion planning. Serverless computing provides a nearly unbounded source of parallelism that we exploit by dividing vertices to connect across lambda functions. Each lambda samples the same vertex sequence by initializing the sampler with a common seed and connects a subset of edges.

In experiments with a Fetch robot, the proposed serverless computing speeds up motion planning computation by up to 52x compared to local baselines. This approach can be used to speed up sporadically computationally-intensive motion-planning problems while being more cost-effective than an always-on high-end computer. Additionally, we provide guidelines for applications to simultaneously optimize for cost and end-time by varying the work-sharing method and the number of lambdas.

In future work, we plan to explore different approaches to sharing information between serverless processes, taking advantage of recent developments in serverless computing [9], to achieve lower startup overhead, faster point-to-point communication, and reduce bottlenecks on scalability.

Acknowledgment

This research was performed at the AUTOLAB at UC Berkeley in affiliation with the Berkeley AI Research (BAIR) Lab, Berkeley Deep Drive (BDD), the Swarm Lab, the Real-Time Intelligent Secure Execution (RISE) Lab, the CITRIS "People and Robots" (CPAR) Initiative, and the NSF ECDI Secure Fog Robotics Project Award 1838833. The information, data, comments, and views detailed herein does not necessarily reflect the endorsements of the sponsors.

REFERENCES

[1] J. Canny, *The complexity of robot motion planning*. MIT press, 1988.

[2] J. Ichnowski, W. Lee, V. Murta, S. Paradis, R. Alterovitz, J. E. Gonzalez, I. Stoica, and K. G. Goldberg, "Fog robotics algorithms for distributed motion planning using lambda serverless computing," in *Proceedings IEEE Int. Conf. Robotics and Automation (ICRA)*, Jun. 2020.

[3] J. M. Hellerstein, J. M. Faleiro, J. E. Gonzalez, J. Schleier-Smith, V. Sreekanti, A. Tumanov, and C. Wu, "Serverless computing: One step forward, two steps back," in *Conference on Innovative Data Systems Research (CIDR '19)*, 1 2019. [Online]. Available: https://arxiv.org/abs/1812.03651

[4] Amazon Web Services, Inc. AWS Lambda – pricing. [Online]. Available: https://aws.amazon.com/lambda/pricing/

[5] N. R. Herbst, S. Kounev, and R. Reussner, "Elasticity in cloud computing: What it is, and what it is not," in *10th International Conference on Autonomic Computing (ICAC 13)*. San Jose, CA: USENIX Association, Jun. 2013, pp. 23–27. [Online]. Available: https://www.usenix.org/conference/icac13/technical-sessions/presentation/herbst

[6] S. Eismann, J. Scheuner, E. van Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. L. Abad, and A. Iosup, "A review of serverless use cases and their characteristics," 2021.

[7] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. M. Carreira, K. Krauth, N. Yadwadkar, J. E. Gonzalez, R. A. Popa, I. Stoica, and D. A. Patterson, "Cloud programming simplified: A berkeley view on serverless computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2019-3, 2 2019. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-3.html

[8] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly, and S. Pallickara, "Serverless computing: An investigation of factors influencing microservice performance," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*, 2018, pp. 159–169.

[9] V. Sreekanti, C. Wu, X. C. Lin, J. Schleier-Smith, J. M. Faleiro, J. E. Gonzalez, J. M. Hellerstein, and A. Tumanov, "Cloudburst: Stateful functions-as-a-service," 2020.

[10] I. E. Akkus, R. Chen, I. Rimac, M. Stein, K. Satzke, A. Beck, P. Aditya, and V. Hilt, "SAND: Towards high-performance serverless computing," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, Jul. 2018, pp. 923–935. [Online]. Available: https://www.usenix.org/conference/atc18/presentation/akkus

[11] A. Mohan, H. Sane, K. Doshi, S. Edupuganti, N. Nayak, and V. Sukhomlinov, "Agile cold starts for scalable serverless," in *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*. Renton, WA: USENIX Association, Jul. 2019. [Online]. Available: https://www.usenix.org/conference/hotcloud19/presentation/mohan

[12] M. S. Aslanpour, A. N. Toosi, C. Cicconetti, B. Javadi, P. Sbarski, D. Taibi, M. Assuncao, S. S. Gill, R. Gaire, and S. Dustdar, "Serverless edge computing: Vision and challenges," in *2021 Australasian Computer Science Week Multiconference*, ser. ACSW '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3437378.3444367

[13] Fetch Robotics, "Fetch research robot," http://fetchrobotics.com/research/.

[14] H. Choset, K. M. Lynch, S. A. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

[15] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high dimensional configuration spaces," *IEEE Trans. Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[16] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001.

[17] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, Jun. 2011.

[18] Y. Li, Z. Littlefield, and K. E. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," *The International Journal of Robotics Research*, vol. 35, no. 5, pp. 528–564, 2016.

[19] N. M. Amato and L. K. Dale, "Probabilistic roadmap methods are embarrassingly parallel," in *Proceedings IEEE Int. Conf. Robotics and Automation (ICRA)*, May 1999, pp. 688–694.

[20] I. A. Şucan and L. E. Kavraki, "Kinodynamic motion planning by interior-exterior cell exploration," in *Algorithmic Foundation of Robotics VIII*. Springer, 2009, pp. 449–464.

[21] J. Ichnowski and R. Alterovitz, "Scalable multicore motion planning using lock-free concurrency," *IEEE Transactions on Robotics*, vol. 30, no. 5, pp. 1123–1136, 2014.

[22] M. Otte and N. Correll, "C-FOREST: Parallel shortest path planning with superlinear speedup," *IEEE Transactions on Robotics*, vol. 29, no. 3, pp. 798–806, 2013.

[23] S. Carpin and E. Pagello, "On parallel RRTs for multi-robot systems," *Proceedings 8th Conference Italian Association for Artificial Intelligence*, 2002.

[24] S. A. Jacobs, N. Stradford, C. Rodriguez, S. Thomas, and N. M. Amato, "A scalable distributed RRT for motion planning," in *Proceedings IEEE Int. Conf. Robotics and Automation (ICRA)*, May 2013, pp. 5073–5080.

[25] M. Kröhnert, R. Grimm, N. Vahrenkamp, and T. Asfour, "Resource-aware motion planning," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 32–39.

[26] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, B. Smith, and R. H. Katz, "Selecting the best VM across multiple public clouds: A data-driven performance modeling approach," in *Proceedings of the 2017 Symposium on Cloud Computing*, ser. SoCC '17. New York, NY, USA: ACM, 9 2017, pp. 452–465. [Online]. Available: http://doi.acm.org/10.1145/3127479.3131614

[27] G. McGrath and P. R. Brenner, "Serverless computing: Design, implementation, and performance," in *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 2017, pp. 405–410.

[28] Amazon Inc., "Aws case studies," https://aws.amazon.com/lambda/resources/customer-case-studies/.

[29] J. Ichnowski, J. Prins, and R. Alterovitz, "The economic case for cloud-based computation for robot motion planning," in *Proceedings International Symposium on Robotics Research (ISRR)*, 2017, pp. 1–7.

[30] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 398–409, 2015.

[31] E. Plaku, K. Bekris, B. Chen, A. Ladd, and L. Kavraki, "Sampling-based roadmap of trees for parallel motion planning,," pp. 597–608, 2005.

[32] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift, "Peeking behind the curtains of serverless platforms," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, Jul. 2018, pp. 133–146. [Online]. Available: https://www.usenix.org/conference/atc18/presentation/wang-liang

[33] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," 1967.

[34] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics and Automation Magazine*, vol. 19, no. 4, pp. 72–82, Dec. 2012. [Online]. Available: http://ompl.kavrakilab.org