A Framework for Self-Adaptive Dispersal of Computing Services

Aaron Paulos*, Soura Dasgupta†, Jacob Beal*, Yuanqiu Mo†, Khoi Hoang‡, J. Bryan Lyles§, Partha Pal*, Richard Schantz*, Jon Schewe*, Ramesh Sitaraman¶, Alexander Wald*, Christabel Wayllace‡, and William Yeoh‡

*Raytheon BBN Technologies, Cambridge, MA 02138

Email: {aaron.paulos, jake.beal, partha.pal, richard.schantz, jon.schewe, alexander.wald}@raytheon.com

†Department of Electrical and Computer Engineering, University of Iowa, Iowa City, IA 52242

Email: {soura-dasgupta, yuanqiu-mo}@uiowa.edu

‡Department of Computer Science and Engineering, Washington University in St. Louis, St. Louis MO 63130

Email: {khoi.hoang, cwayllace, wyeoh}@wustl.edu

§University of Tennessee Knoxville, Knoxville, Tennessee 37996

Email: jblyles@acm.org

¶College of Information and Computer Sciences, University of Massachusetts, Amherst, MA 01003

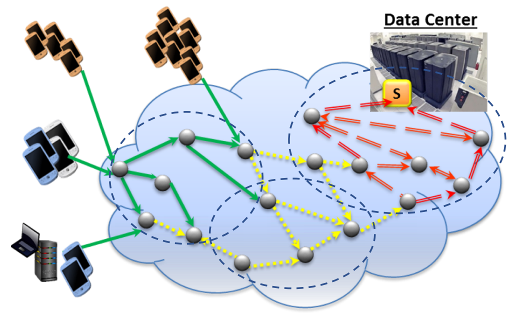
Email: ramesh@cs.umass.edu

***Abstract*—Modern networking architectures are making it increasingly possible to disperse services not just across servers but into intermediate network devices as well. Here we introduce the Misson-oriented Adaptive Placement (MAP) architecture, which synthesizes prior work on middleware, load-balancing, constraint solving, and aggregate programming into a framework for self-adaptive management of dispersed services. We provide a first evaluation of the efficacy and resilience that can be provided through this approach: results in simulation demonstrating that MAP can autonomously change the deployment of services to adapt to changing needs and failures.**

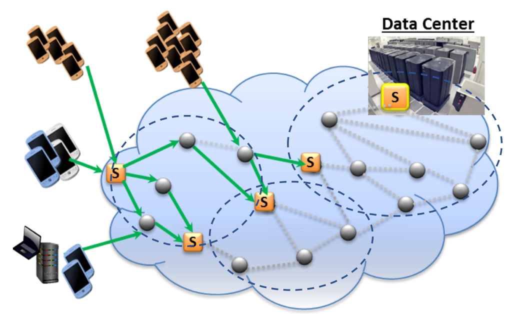
***Index Terms*—dispersed computing, middleware, load-balancing, aggregate programming**

I. Introduction

To accommodate highly interactive and real-time applications, computational resources are being dispersed into the network. Computing near the user enables highly responsive and interactive applications like real-time control of devices, computing-in-the-loop decision making, and graphics. Moreover, modern networking hardware can not only route but also perform general high-speed programmable information processing. This can enable new frameworks of distributed computing in which services run not just at user or server machines, but are dispersed across a wide variety of appliances in the network. In contrast, current service architectures concentrate elasticity in data centers (Figure 1), where computational power and storage abound but network and timing constraints are limiting. Exploiting "hidden hardware" by dispersing services into the network can make services faster and more resilient.



(a) Data center services



(b) Dispersed computing

Fig. 1. Current service architectures concentrate elasticity in data centers (a), leaving services vulnerable to network and timing constraints, particularly across heterogeneous sub-networks. Bottlenecks can be avoided by dispersing computation to network hardware (b).

To this end, we propose Mission-oriented Adaptive Placement (MAP), a distributed, multi-layer framework for decision making and resource management. MAP combines prioritized management of in-network compute/storage resources with adaptive placement and migration of computing tasks, in

Supported by Defense Advanced Research Projects Agency (DARPA) contract HR001117C0049. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. This document does not contain technology or technical data controlled under either U.S. International Traffic in Arms Regulation or U.S. Export Administration Regulations. Approved for public release, distribution unlimited (DARPA DISTAR 31059, 3/18/19).

response to changes in demand, availability, and load of network and computing resources. MAP brings together prior advances in middleware and container technology (giving services platform independence and mobility), load balancing and distributed constraint optimization (to allocate services into sub-networks and individual devices) and aggregate programming (AP) (for self-stabilizing information summarizing, dispersal, and system integration).

In the remainder of this paper, we introduce the MAP architecture for dispersed computing and provide a first evaluation of its efficacy and resilience. Section II reviews the context and related work, Section III presents the MAP architecture, Section IV empirically validates the self-adaptation and resilience of MAP, and Section V summarizes contributions and outlines future work.

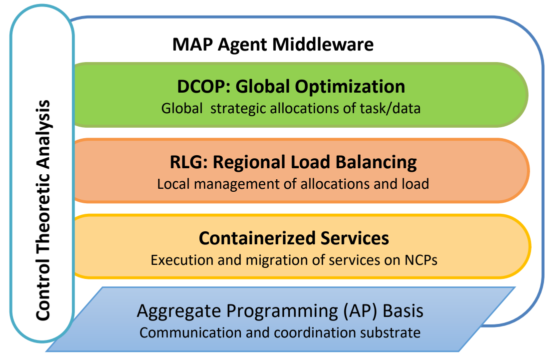


Fig. 2. High-level architecture of MAP: global optimization by DCOP allocates tasks and data across regions at a low update frequency, setting constraints on each region's implementation of high-frequency load balancing by RLG across containerized service instances. All of these are connected using aggregate programming (AP) primitives for resilient communication and coordination and analyzed for stability both individual and collectively.

II. Problem Context and Related Work

Middleware injects abstraction layers between application software and underlying operating systems, hardware, and networks [1], which can be used to address many distributed computation issues. Examples like RMI [2], CORBA [3], DDS [4], and various ESBs [5] offer different interaction paradigms ranging from remote procedure calls (RPC) and synchronous/asynchronous messaging to publish-subscribe and representational state transfer (REST) [6], and support value adds like authentication, fault tolerance, and management of cache, network bandwidth, and OS priorities [7]. While most middleware facilitates use of system resources or remote services, some also enable adaptations such as changing the functional behavior of the application, how resources are used, or the number and location of distributed components used [8], [9]. More recently, as cloud computing begins to migrate compute and storage capacity into the network and toward the network edge (edge computing and fog computing), middleware is being used to perform additional tasks like aggregation and staging, accounting and tracking resource use, and security (for edge sensors and IoTs that do not speak IP or lack enough hardware/power) [10]–[12]. The conception of MAP draws directly on these antecedents.

Load balancing and resource allocation have been extensively studied for distributed hosting of applications and services. Various mechanisms for load balancing of web applications are outlined in [13], including a DNS-based approach used by MAP. Load balancing has also been widely used in cloud systems for VM placement, consolidation, and migration to better utilize cloud resources [14]. At large scale, load balancing often adopts a layered approach, in which different strategies are used for balancing load globally across regions and locally within a region. For instance, the Akamai network [15], which hosts a significant fraction of all web and video applications, uses the stable marriage algorithm as a global load balancer allocating resources across regions [16] and consistent hashing for load balancing within a region [17]. MAP draws on these as well, making use of the same cloud VM technologies and adopting a two-layer approach derived from that of Akamai, but adapting them for a highly dispersed network rather than a data-center oriented architecture.

MAP also departs from most prior load balancing work in its goals. Whereas most prior work aims at a best-effort allocation of resources to maximize resource utilization, MAP instead aims for optimal satisfaction of a set of mission priorities. To this end, MAP formulates global load balancing as a Distributed Constraint Optimization Problem (DCOP), in which a group of agents need to coordinate their value assignments to minimize the sum of the resulting constraint costs [18]–[20]. This model has been used to solve several multi-agent coordination problems including distributed meeting scheduling [21], sensor/robot coordination [22], coalition formation [23], smart grids [24], and smart homes [25]. Closer still is [26], which applies DCOP to dynamic load balancing, although that work addresses migration of wireless load sources, rather than services. Our layered approach mitigates scaling challenges faced by these prior systems, which formulate all load balancing as a single DCOP problem, whereas MAP optimizes only at the global level where the number of regions is tractable.

III. The MAP Framework

The MAP framework is designed to operate on backhaul networks that connect between pools of client devices and one or more data centers (as in the example diagram in Figure 1), as are often deployed by emergency response, military, or large government or commercial organizations. Many nodes in this network are general-purpose hardware capable of hosting services and a MAP agent. We call them Networked Computation Points (NCPs); in this paper, we assume that all non-client devices are NCPs.

MAP is designed to adaptively place and migrate services (and associated data) that are currently consolidated in data centers into the in-network NCPs for data-centric applications with a high degree of localization and structure (e.g., publish-subscribe applications, aggregation and filtering of sensor reports, or processing of imagery and video). MAP further organizes the NCPs into regions, both to handle the expected scale of significant real-world deployments and to follow existing structural or organizational divisions in the network.

To meet these goals, the MAP framework is organized into the three-layer high-level architecture shown in Figure 2. At the bottom, every NCP in the network hosts a MAP agent and some number of containerized services, with the agent monitoring and managing NCP resource usage, and executing and migrating services. In the middle, one MAP agent per region is elected as its Regional Load-Balancing Gateway (RLG) and executes an algorithm to manage allocations and load across NCPs within a region, deciding how many containers of each service will be hosted at each NCP (by starting and stopping containers) and using DNS updates to route client requests to NCPs. At the top, all RLGs participate in a DCOP algorithm, which at a much slower rate allocates services to regions.

For stable and resilient interaction across NCPs, regions, and layers, MAP uses aggregate programming (AP) as the communication and coordination framework connecting all MAP agents at every layer, providing self-adaptive coordination amenable to controls analysis for stability and convergence. AP was developed to improve prediction and composition of distributed systems [27], founded on the *field calculus* computational model [28], [29], which provides a dual semantics for both collective system behavior and the individual asynchronous actions single devices take in order to produce that behavior. In particular, MAP uses the self-stabilizing AP "building blocks" introduced in [30]: G blocks spread information and C blocks summarize information. This ensures resilience in communication and enables control-theoretic analysis of MAP. These AP building blocks maintain network state estimates at every RLG (and thus also the DCOP algorithm) at the beginning of each round (comprising the service demand arriving in the region and the load and allocated containers for each service on all NCPs), and also carry communications that disseminate supporting information, such as service descriptions.

DCOP is used to periodically optimize region-level adaptive placements (the region level is more tractable for computationally expensive multi-objective network optimization). Load-balancing and resource management within each region can then permit faster provisioning and dispatching decisions within the bounds set forth by the overarching DCOP solution, but working independently and in parallel to it. The global DCOP, the regional load-balancing, the execution of containerized services on NCPs, and the AP "glue" thus form an interacting self-adaptive dynamical distributed system.

Realizing the MAP framework also addresses a number of secondary engineering and integration issues. To facilitate transparent deployment, the framework uses existing DNS name resolution services to direct client requests to services hosted in an appropriately configured virtualization and container architecture. MAP also relies on existing authentication and encryption techniques, and on other off-the-shelf protection and recovery methods to aid with security and resilience.

At this global level, the expected effects of MAP are as follows: without MAP, client requests are sent to the data center, and in response to increased load or demand additional resources or servers are commissioned in one or more data centers for load balancing. Allocations are confined to data centers relying on backhaul access. Dependency or load on the backhaul does not decline and even increases. Failing links or attacks on the backhaul make the service delivery degrade or fail. Failure, congestion and traversal delay are often fatal to remote user expectations, especially in time-critical settings.

MAP can improve the operation and use of services in several ways. Upon initialization, MAP's multi-layer decision making algorithms may pre-position some tasks and data at selected NCPs depending on mission needs or application affinity. Directing services to NCPs, instead of the data center, reduces backhaul traffic and response time. With or without prepositioning, MAP monitors the use of applications, and periodically assigns and adjusts the in-network resources for specific services, gradually migrating services to regions nearer to where the demands are and maximizing the use of available resources, yet again reducing backhaul traffic and response time. Moreover, by enabling services to run in diverse locations, MAP reduces dependencies on specific links and NCPs, particularly those close to the datacenter, making applications more resilient to unplanned events.

The modular design of MAP also enables improvement of the system by upgrades to individual components, such as the particular DCOP or RLG algorithm, the containerization method, or individual service applications. The initial DCOP algorithm has been described in [31], and we describe the heuristics of a simple RLG below.

A. Heuristics of a simple RLG

RLG must balance allocation of containers against unknown future demand, given that some services can take a long time to handle arriving demands. Allocation of too few containers may cause job drops due to overloading; allocation of too many may under-load containers, which may be "stuck" completing long jobs while other service jobs lack available containers.

We now present a simple reactive load-balancing on homogeneous NCPs with a few obvious alternative rules, to establish a baseline for performance and highlight challenges for any RLG implementation. In MAP each region has multiple NCPs, and each with multiple containers. This simple RLG algorithm allocates a fixed number (N) of containers to a service if the estimated load exceeds a proportion, p^+ of the total capacity assigned to that service. Each container to start is allocated to an NCP following one of several heuristics: (i) most available containers (MAC), (ii) least load percentage (LLC), or (iii) already running the service now (SN), and either: (a) all new containers started on the same NCP (bangbang), or (b) spread evenly across NCPs (smooth). Complementarily, one container at a time stops when the estimated load falls below a fraction $p^- < p^+$, of the capacity allocated to the service. Once this threshold is met, stopping occurs after waiting t_{delay} time.

Competing needs guide the choice of parameters. Stopping too slowly due to low p^- or t_{delay} deprives needy services of containers. As a corollary, stopping multiple containers at

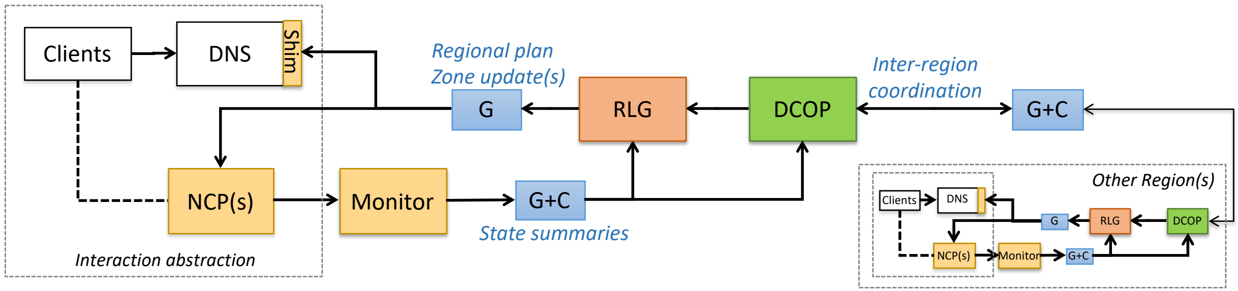


Fig. 3. Control flow of MAP communication and decision-making across MAP elements (colored) hosted on NCPs and DNS nodes of network components. Aggregate programming primitives (blue) resiliently summarize information from monitors to the RLG (orange) and DCOP (green) decision-making agents, as well as disseminating information for inter-region coordination between DCOP and communicating the decisions of DCOP and RLG.

a time is desirable, though this can also cause job drops. Too large a p^+ may cause job drops by depriving needed growth space for services with high loads. Slow processing can cause job drops or persistent starting and stopping, while fast processing can induce hysteresis in container allocation: an initially oversubscribed service whose demand drops receives more containers than one with initially few containers, even if the steady state demands are identical.

Among the three heuristics, MAC starts more containers at a time than SN and LLC and is less likely to have job drops. MAC also stops containers faster with demand drop, but may cause a service to retain more containers than needed. Smooth initial allocations withstand higher demands over bang-bang, i.e., initially distributing services over many NCPs is preferable.

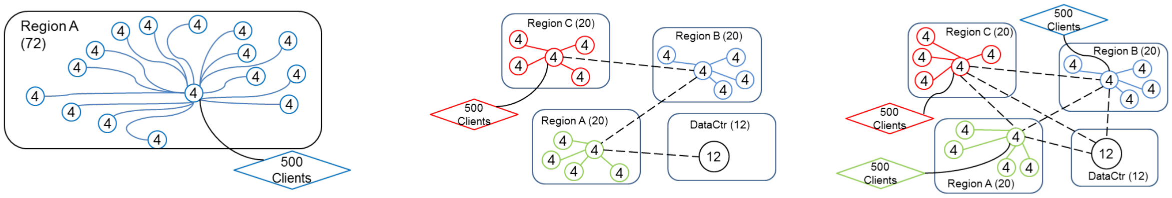
Finally, we note that parameters like p^-, p^+, the number of containers started at a time, and the number stopped at a time are fixed. We believe extension to an adaptive approach is desirable, where parameters can change according to system state and predicted demand, e.g., allocating more containers to meet a surge in predicted demand, or raising p^- to deal with a service with too many containers.

IV. Experimental Evaluation

To evaluate the MAP framework we have prototyped a lightweight network and compute emulation capable of hosting simulated services. The emulatation extends the framework presented in [32], and pending public release approval, the emulation, MAP code, and experimental scenarios supporting these experiments (including a full listing to reproduce this paper's results) will be publicly available online at https://github.com/map-dcomp. Here we report evaluation of the efficacy of MAP for self-adaptation by comparing scenarios in which MAP disperses processes against scenarios where elasticity is confined to a datacenter. We also evaluate the resilience of MAP against node failures. All experiments are run on dedicated machines with Xeon E3 4-core 3.1 GHz processors, 32 GB of RAM, and Ubuntu 18.04.

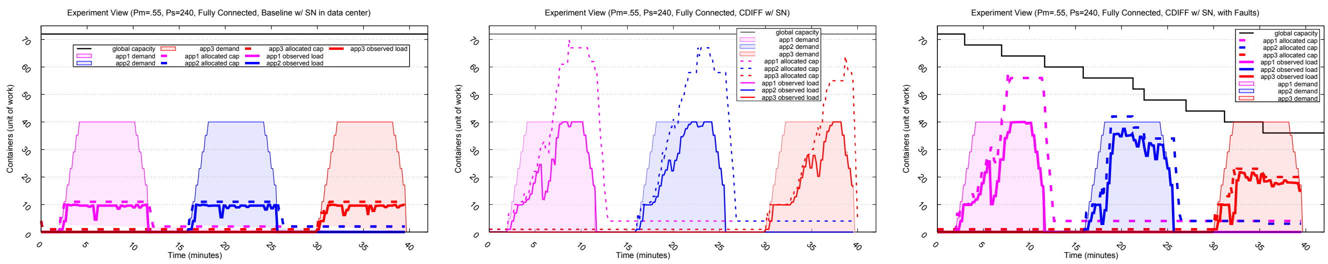
As test networks we consider 16 or 17 NCPs either all in a single region or partitioned into three regions with a separate datacenter, where multi-region configurations form either a chain or fully connected topology (Figure 4). We refer to these networks hereafter as Single, Chain or Full. From the edge of these networks, pools of 500 clients send requests for one of three services in a shifting pattern of demand (for Full we use three pools), where demand rises and falls sharply for each service in turn, thus shifting the resource allocation demands on the system over time. For these initial tests, we consider abstracted versions of compute- and memory-intensive processes (e.g., searching for designated objects in an image stream) as our example services, assume that each NCP can be further partitioned into four service containers, the data-center can run 12 containers, and that network capacity is non-limiting. As the processes modeled are compute-intensive, once initiated, each served request run a job on a container for just over 20 seconds before completing. In all tests, the MAP implementation runs an agent on every NCP with a fixed leader in each region where AP is run every 0.5 seconds, RLG is run every 3 seconds (using parameters $p^+ = 0.75$, $p^- = 0.25$, and $t_{delay} = 3$), and DCOP is run with a 60 second delay between the end of one cycle and the beginning of the next cycle.

Figure 5 shows time trace behavior of shifting client demand for 40 containers without MAP, with MAP, and with NCP-failures and MAP. Each trace is based upon a Full topology, where the data center is under-provisioned to handle the incoming client demand. The traces and scenarios illustrate MAP's benefit over the baseline. They also illustate how the MAP prototype responds to NCP conditions, including failures. Across all figures, the shaded areas show the total experimental demand induced by the edge clients, where the three color codings represent demands for the services *app1*, *app2*, and *app3*, following a demand flash profile. The maximum compute capacity for the network is represented in task containers as the black line, where dips in the third trace are pre-determined, uniformly randomized NCP failures across time. The solid red, blue, and magenta lines show the total load that is successfully serviced, i.e., the sum of load from each client requests that was able to fit within the available processing capacity of the container to which it was dispatched by DNS. Individual client requests that do not fit within a target active container are considered a failed request for our simulation. As a result, the shaded area above a solid load line can be thought of as requests that were not serviced by active containers. The dashed lines show the capacity allocated for



(a) Single (b) Chain (c) Fully Connected

Fig. 4. Illustration of networks under test: (a) single region with 500 clients, (b) chain of regions with data center on one end and 500 clients on the other end, and (c) three regions and data center in fully connected topology, with 500 clients at each region.



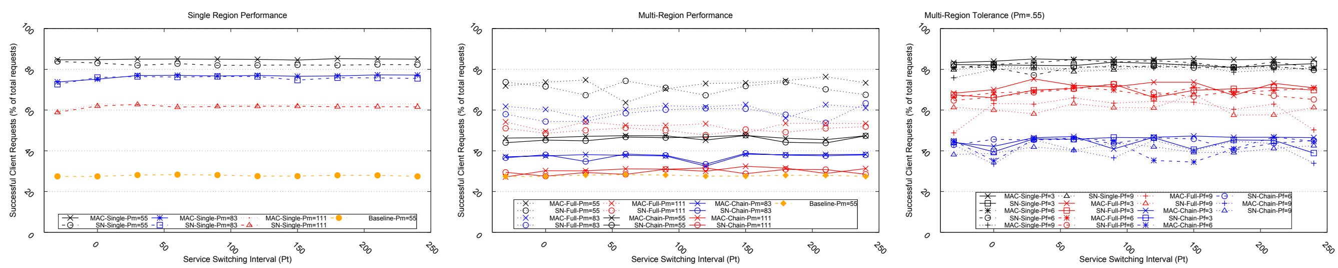
(a) Trace without MAP (b) Trace with MAP (c) Trace with MAP and NCP Failures

Fig. 5. Experiment traces vs. time (a) without MAP, (b) with MAP, and (c) MAP with failures scenarios for Full topology, Pm=0.55, Pt=240 and Pf=9. Shaded areas show experimental client demand, solid lines show observed load, and dashed lines show allocated capacity capable of processing demand.

each service, as the number of containers that are dynamically instantiated to handle load. For the "with MAP" figures, the allocation at any point and time is a direct result of the CDIFF DCOP and SN RLG implementations, where the difference between the solid load line and the dotted line within the shaded area can be thought of as the client load that could be processed if load-balanced into some container with capacity. Finally, for all baseline experiments, without MAP, we apply only the SN algorithm to simulate vertical data center scaling.

Examining the collection of trace behaviors in these illustrations, we observe that dispersing services into a network with more capacity, even in face of failures, should always perform better than keeping those services in the datacenter. This intuitive observation is a result of the fact that the total network compute capacity has a greater capacity than just the data center alone. In the MAP-only trace, we see unprocessed demand where the rate of new service requests is rising faster than the system can start new containers to serve them. This outcome is a direct result of the sequential allocation of new containers to NCPs by the RLG (i.e., an artifact of our initial simple algorithm), and being that this is a Full topology, CDIFF will also need time to react to the demand and spread services out into peer-regions with available capacity. The overfitting of the allocation scheme can be explained as a result of the greedy RLG allocation scheme and the rapid stopping of each service is an outcome of the parallized stop container operation in the RLG. In the MAP with failures figure, we trace MAP's tolerance to 9 NCP failures, i.e., removing 36 containers of capacity. Across this run, we see the effect of failures in both the second application curve, where two NCPs hosting traffic fail in rapid succession, and in the third application curve, where resources are significantly depleted. In the former case, MAP's DNS load-balancer, part of the RLG, is effectively handling demand by spreading requests. In the latter case, the simple algorithms in our implementation are only able to partially adapt, but the performance is still well above the non-MAP baseline.

Figure 6 summarizes the overall performance of MAP as we vary the stress on the system. We vary the following: (Pt) the time interval between halting demand for one service and starting the next ranging from -30 seconds to 240 seconds to test adaptation dynamics; (Pm) the total client demand magnitude as 55%, 83%, 111% against the 72 container network compute capacity (i.e., 40, 60, and 80 containers of demand); (Pn) the network topology as Single, Chain, or Full; (Pf) as 3, 6, or 9 faults; (Ph) as the RLG heuristic types SN or MAC. For baseline experiments without MAP, we use the Chain configuration. Figure 6(a) assesses the efficacy of self-adaptation as a function of the rate at which demand shifts from one service to another. As can be seen, even for MAP is always able to adapt quickly enough within its, only even slightly degrading as the demands begin to overlap, and always well above datacenter-only performance. Not also that, as predicted, MAC is always equal to or better than SN. Multi-region performance is lower than single-region performance (Figure 6(b)), due to the requirements for DCOP as well as RLG, but still far above the data center baseline. Figure 6(c) assesses the resilience of MAP to failures, showing that MAP's performance degrades gracefully with the rate of failure: self-monitoring allows services to shift from failed nodes to surviving nodes, though the loss of capacity means that not all demand can be shifted. In summary, our simulations show that MAP is able to fulfill the goal of increasing self-adaptation and resilience through dispersal of services into NCPs.



(a) Shifting Demand Single Region (b) Shifting Demand Multi Region (c) NCP Failures

Fig. 6. Experimental response to perturbations, illustrating (a) single-region and (b) multi-region efficacy and (c) multi-region tolerance to NCP failure scenarios, where Pt, Pm, Pn, Pf, and Ph are varied. Each point shows client success rates. In all cases, MAP shows improvement over datacenter-only.

V. Contributions

We have shown that MAP can disperse services out of data centers into intermediate NCPs throughout a network, improving performance and resilience of networked services. Future work will aim to increase the scale and realism of networks used to validate the MAP framework, and to begin transition into operational use in real-world systems, including improving RLG and DCOP algorithms, as well as handling service migration, dependencies, and decomposition. MAP may also benefit from using advanced networking techniques, like stochastic or multi-path routing. Finally, there are a number of potential opportunities for application of the core ideas in MAP and its components to other aspects of self-managing and resilient networked computing systems.

References

[1] D. Schmidt, "Middleware for Real-time and Embedded Systems," *Communications of the ACM*, vol. 45, no. 6, pp. 43–48, 2002.

[2] Oracle, "Java RMI," https://docs.oracle.com/javase/tutorial/rmi/index.html, Retrieved 2019.

[3] Object Management Group, "Common Object Request Broker Architecture (CORBA)," https://www.omg.org/spec/CORBA, Retrieved 2019.

[4] ——, "Data Distribution Service," https://www.omg.org/omg-dds-portal/, Retrieved 2019.

[5] D. Chappell, *Enterprise Service Bus*. OReilly, 2004.

[6] R. T. Fielding, *Architectural Styles and the Design of Network based Software Architectures*. PhD Thesis, UC Irvine, 2000.

[7] E. Nygren, R. K. Sitaraman, and J. Sun, "The Akamai Network: A Platform for High-Performance Internet Applications," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 3, pp. 2–19, 2010.

[8] J. Zinky, D. E. Bakken, and R. E. Schantz, "Architectural Support forQuality of Service for CORBAObjects," *Theory and Practice of Object Systems*, vol. 3, no. 1, pp. 55–73, 1997.

[9] Y. Ren, D. E. Bakken, T. Courtney, M. Cukier, D. A. Karr, P. Rubel, C. Sabnis, W. H. Sanders, R. E. Schantz, and M. Seri, "AQuA: An Adaptive Architecture that Provides Dependable Distributed Objects," *IEEE Transactions on Computers*, vol. 52, no. 1, 2003.

[10] M. Iorgam, L. Feldman, R. Barton, M. J. Martin, N. Goren, and C. Mahmoudi, "Fog Computing Conceptual Model, Recommendations of the National Institute of Standards and Technology," NIST Special publication 500-325, 2018.

[11] "Openfog consortium," https://www.openfogconsortium.org/.

[12] S. Hachem, V. Issareny, V. Mallet, A. Pathak, and P. Bhatia, R. amd Raverdy, "Urban Civics: An IoT Middleware for Democratizing Crowdsensed Data in Smart Societies," in *Research and Tech. for Society and Industry Leveraging a Better Tomorrow (RTSI)*, 2015, pp. 117–126.

[13] V. Cardellini, M. Colajanni, and P. S. Yu, "Dynamic load balancing on web-server systems," *IEEE Internet computing*, vol. 3, no. 3, pp. 28–39, 1999.

[14] M. Xu, W. Tian, and R. Buyya, "A survey on load balancing algorithms for virtual machines placement in cloud computing," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 12, p. e4123, 2017.

[15] E. Nygren, R. Sitaraman, and J. Sun, "The Akamai Network: A platform for high-performance Internet applications," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 3, pp. 2–19, 2010.

[16] B. M. Maggs and R. K. Sitaraman, "Algorithmic nuggets in content delivery," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 3, pp. 52–66, Jul. 2015.

[17] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *ACM STOC*, 1997, pp. 654–663.

[18] P. Modi, W.-M. Shen, M. Tambe, and M. Yokoo, "ADOPT: Asynchronous distributed constraint optimization with quality guarantees," *AIJ*, vol. 161, no. 1–2, pp. 149–180, 2005.

[19] A. Petcu and B. Faltings, "A scalable method for multiagent constraint optimization," in *IJCAI*, 2005, pp. 1413–1420.

[20] F. Fioretto, E. Pontelli, and W. Yeoh, "Distributed constraint optimization problems and applications: A survey," *JAIR*, vol. 61, pp. 623–698, 2018.

[21] R. Maheswaran, M. Tambe, E. Bowring, J. Pearce, and P. Varakantham, "Taking DCOP to the real world: Efficient complete solutions for distributed event scheduling," in *AAMAS*, 2004, pp. 310–317.

[22] F. Delle Fave, A. Rogers, Z. Xu, S. Sukkarieh, and N. Jennings, "Deploying the max-sum algorithm for decentralised coordination and task allocation of unmanned aerial vehicles for live aerial imagery collection," in *ICRA*, 2012, pp. 469–476.

[23] S. Ueda, A. Iwasaki, M. Yokoo, M. Silaghi, K. Hirayama, and T. Matsui, "Coalition structure generation based on distributed constraint optimization," in *AAAI*, 2010, pp. 197–203.

[24] F. Fioretto, W. Yeoh, E. Pontelli, Y. Ma, and S. Ranade, "A distributed constraint optimization (DCOP) approach to the economic dispatch with demand response," in *AAMAS*, 2017, pp. 999–1007.

[25] F. Fioretto, W. Yeoh, and E. Pontelli, "A multiagent system approach to scheduling devices in smart homes," in *AAMAS*, 2017, pp. 981–989.

[26] S. Cheng, A. Raja, J. Xie, and I. Howitt, "DLB-SDPOP: A multiagent pseudo-tree repair algorithm for load balancing in WLANs," in *Intelligent Agent Technology*, 2010, pp. 311–318.

[27] J. Beal, D. Pianini, and M. Viroli, "Aggregate programming for the internet of things," *IEEE Computer*, vol. 48, no. 9, pp. 22–30, 2015.

[28] F. Damiani, M. Viroli, and J. Beal, "A type-sound calculus of computational fields," *Science of Computer Programming*, vol. 117, pp. 17 – 44, 2016.

[29] G. Audrito, M. Viroli, F. Damiani, D. Pianini, and J. Beal, "A higher-order calculus of computational fields," *ACM Transactions on Computational Logic (TOCL)*, vol. 20, no. 1, p. 5, 2019.

[30] J. Beal and M. Viroli, "Building blocks for aggregate programming of self-organising applications," in *IEEE SASO Workshops*, 2014, pp. 8–13.

[31] K. D. Hoang, C. Wayllace, W. Yeoh, J. Beal, S. Dasgupta, Y. Mo, A. Paulos, and J. Schewe, "New distributed constraint satisfaction algorithms for load balancing in edge computing: A feasibility study," in *10th International Workshop on Optimization in Multiagent Systems (OptMAS)*, May 2019.

[32] S. S. Clark, J. Beal, and P. Pal, "Distributed recovery for enterprise services," in *IEEE SASO*, Sept 2015, pp. 111–120.