Research Statement

Jayaram Kallapalayam Radhakrishnan (K. R. Jayaram)

http://www.jayaramkr.com

I am passionate about designing and implementing large-scale distributed systems, and empirically measuring and analyzing their performance. This is because they present a unique set of challenges, arising from the need to support customer-facing, highly-available services and applications, while scaling to thousands of servers and network components possibly running on multiple geographically distributed data centers. Engineering *effective* distributed systems is a hard problem. Research in this area has the potential to impact several emerging applications ranging from electronic commerce, financial trading, advertising and marketing to social networks and sensor networks. Moreover, with the increasing use of mobile devices, a large percentage of software is becoming distributed or so-called "cloud-backed software"/"apps". Such applications need to seamlessly execute on multiple devices while maintaining durability and consistency of data – well-known examples are text editing, spreadsheet and presentation software.

Research Objective and Methodology

It is very hard for distributed systems developers to achieve scalability *and* efficiency. The goal of my research is to *make engineering scalable and efficient distributed systems easier*. One way to identify inefficiencies and scalability limitations/bottlenecks in existing distributed systems is through empirical analysis. Once I identify such bottlenecks, I start by developing fundamental building blocks and corresponding programming abstractions to alleviate them. I then align these abstractions with other programming language techniques, middleware systems and distributed algorithms to increase scalability and efficiency.

My research methodology can thus be summarized as: (1) identifying inefficiencies and scalability limitations of existing distributed applications and algorithms, (2) formally defining distributed programming abstractions; specifying their feasible and desirable guarantees, (3) devising and prototyping effective algorithms to realize these abstractions, (4) using analytical models to characterize the performance of the algorithms, and (5) designing large scale experiments to empirically measure their performance under realistic workloads. I have found that building a working system and performing realistic experiments typically reveals new challenges and performance considerations that otherwise might have been ignored.

One common theme among my of research projects is to explore and exploit potential synergies between programming languages and and middleware systems to increase the efficacy of distributed applications. This is vital to emerging distributed applications like algorithmic trading, online advertising, etc. My PhD thesis [2, 5, 8, 9, 7] describes how the scalability and efficiency of event-based distributed applications can be increased by aligning middleware systems with specialized programming abstractions. This research has been recognized by a *best paper award at MIDDLEWARE 2010* and by the Purdue University Computer Science department though the *Maurice H. Halstead award*. Effective application-middleware interaction is also vital to cloud computing applications for *automatic* elastic scalability – to be able to request/relinquish additional computational resources from cloud management middleware in response to increasing load.

I have extensive experience designing, building and working with *a variety of* real large-scale distributed systems, and this experience has defined my approach and vision for distributed systems research. I have used Marketcetera, an open-source algorithmic trading platform to benchmark the techniques developed during my PhD. My postdoctoral research at HP Labs has given me extensive experience in working with programmable elasticity through elastic remote methods [3], a large-scale transactional distributed graph store called Concerto [11], an intrusion detection system based on event-correlation called ArcSight and distributed shared memory-like abstractions supporting efficient low-latency transactions. During my internship with IBM Research, I've worked on hypervisor-level content-aware caching of virtual machine images to increase the runtime performance of VM instances in clouds that stream portions of VM images on demand [10, 1]. I've also worked on the design and implementation of an elastic event matching system using distributed hash tables for Amazon.com's Simple WorkFlow (SWF) web service. My postdoctoral research experience, my PhD dissertation and my internships have given me an extensive background in distributed systems, and I'm seeking a research position that helps me leverage this background to be productive professionally.

Programmable Elasticity

Elasticity is a key advantage of cloud computing or utility computing, because it enables distributed applications to scale up/down *on demand* in response to changing workloads. Existing distributed applications cannot take advantage of the cloud without elasticity being retrofitted to them. In fact, to optimize the performance of new or existing distributed applications while deploying or moving them to the cloud, engineering robust elasticity management components is essential. This is especially vital for applications that do not fit the programming model of *implicitly* elastic frameworks like Hadoop, Pig, etc., but require high performance (high throughput and low latency), scalability and elasticity – the best example is the class of *datacenter infrastructure applications* like key-value stores (e.g., memcached, HyperDex), consensus protocols (e.g., Paxos), distributed lock managers (e.g., Google's Chubby) and message queues.

Elasticity frameworks which rely on externally observable resource utilization metrics (CPU, RAM, etc.) are insufficient for such applications due to their inherent complexity. We therefore need programming frameworks for *flexible elastic scaling*, that allow developers to build elasticity management components right into the applications by using a combination of resource utilization metrics and fine-grained application-specific information like the properties of internal data structures to drive scaling decisions. This enables the developer to have increased control over the point at which an application is scaled up/down and also encourages auto scalability to be incorporated early in the software development process.

Elastic Remote Methods. To this end, I have developed ElasticRMI [3], a distributed programming framework and middleware system that (1) enables application developers to dynamically change the number of (server) objects available to handle remote method invocations with respect to the application's workload, without requiring changes to clients (invokers) of remote methods, (2) enables flexible elastic scaling, and (3) provides a generic, high-level programming framework that handles elasticity at the level of classes and objects, masking low-level platform specific tasks (like creating and provisioning virtual machine (VM) images) from the developer. Our experiments with four real-world applications implemented in ElasticRMI demonstrate that using fine-grained application-specific metrics actually helps, and increases elasticity significantly. ElasticRMI is an example of my research philosophy, where distributed programming abstractions and runtime/middleware systems are co-designed with the objective of addressing elastic scalability. ElasticRMI's runtime/middleware system uses Apache Mesos for the dynamic instantiattion of new server objects on different nodes in a datacenter/public cloud.

Event-based Distributed Systems

Event-based design of distributed systems is appealing because it *decouples* software components and enables the distributed system to scale to a large number of components. Apart from being an effective way to engineer distributed applications, *extreme event processing* and *event correlation* are key aspects of "big data" research. Handling (collecting, aggregating, correlating and reacting to) streams of events (data) is a key aspect of modern "big-data" applications including social networks (e.g., Twitter, Facebook), algorithmic high frequency trading, intrusion detection, data center monitoring, and sensor network based applications like highway traffic monitoring, smart buildings and smart cities. In my PhD dissertation, I identify and analyze inefficiencies in emerging event-based distributed applications and demonstrate how they can be overcome using specialized programming abstractions and program analyses for event correlation [2, 6], parametric subscriptions [9] and subscription splitting and normalization [7].

Parametric Publish/Subscribe [9, 8]: Event-based distributed systems using publish/subscribe middleware systems are increasingly used in dynamic application scenarios, e.g., in high frequency algorithmic trading to analyze financial data to make effective trading decisions. Here, there is a need to constantly adapt their subscriptions in response to changing market conditions. Under a high frequency of subscription adaptations, re-subscriptions, which are the conventional way to adapt subscriptions, cause content-based publish/subscribe (CPS) systems to spend more resources processing re-subscriptions than routing events between components. To mitigate these problems, I introduce a new programming abstraction called *parametric subscriptions*[9, 8], which are subscriptions where event attributes are compared to a subscriber's internal variables. To support parametric subscriptions in CPS systems, I have designed and implemented efficient algorithms to propagate subscriber variables into the brokers (event routers) in a CPS system. To align the parametric subscriptions abstraction with CPS systems, I have also designed static analyses [6] to automatically identify subscriber variables used in parametric subscriptions, and instrument the application to automatically update the CPS system when the values of these variables change. My work on parametric subscriptions is an ideal example of synergy between distributed applications and middleware systems, where a new abstraction is used to change the interaction between them and the performance of the entire system improves significantly.

To avoid global variable references and to make handling parametric subscriptions fault-tolerant, I have introduced the novel concept of broker variables [9, 8] and variable renaming to retain the decentralized nature of CPS middleware systems. To prove that the parametric subscriptions abstraction is widely applicable, we have implemented it in two CPS middleware systems – the widely cited Siena CPS system and EV, our own CPS system based on the Rete algorithm. We have empirically demonstrated the efficacy of our implementations on multiple benchmarks with different characteristics, using the metrics of throughput, latency, delay and the frequency of spurious events.

Effective Event Correlation through EventJava: EventJava [2] is an extension to Java for generic event-based distributed programming. It provides key programming abstractions that allow application developers to focus on defining, producing and reacting to events, leaving the compiler responsible for generating efficient code for unicasting/multicasting events, detecting event correlation patterns and dispatching event handlers in response to these patterns. The key goal of EventJava is to make event-based programs scalable and efficient. For efficient correlation of events, the EventJava compiler uses definitions of complex events to generate an optimized complex event detection component, called GenTrie. GenTrie (*Gen*eralized *Trie*) [4] consists of an event flow graph, that is designed to discard unwanted events early and to effectively store events that partially match correlation patterns.

Event correlation is also concerned with identifying causal relationships between events. In the absence of synchronized clocks, causal order between events, especially when these are broadcast, has traditionally been achieved by manually mapping events to "broadcast groups", implemented by means of dedicated protocols, to capture dependencies. This is not only tedious for programmers (groups may overlap) but unsafe as dependencies are easily overlooked. Pessimistically funneling all events – even those requiring no ordering with respect to other event types and each other – through a same group strongly hampers efficiency. Our causality analysis of EventJava programs [6] aligns programs with group communication systems like JGroups. It infers all possible dependencies (for safety) from programs, and with hints from the programmer on independencies (for efficiency), allows for *multiple* multicast groups to be be created adequately and manipulated automatically by the runtime [6]. We have demonstrated empirically that this increases the scalability of event-based distributed applications [6].

Subscription Splitting and Normalization for Effective Event Propagation: We have had further success in using program transformations to increase the scalability of event-based distributed systems. While evaluating EventJava programs using CPS middleware systems like Siena, I observed that the use of a single data structure to store subscriptions at brokers created performance bottlenecks. This is because many existing matching algorithms for CPS systems have time complexity linear in the number of subscriptions stored at the broker, i.e. $O(N)$, where N is the number of subscriptions. To reduce the complexity of event-processing at a broker, our EventJava compiler normalizes subscriptions and rewrites them as constraints on intervals and sets. Our Beretta CPS system [7] splits normalized subscriptions by event type and then by the name/type of an attribute. Splitting enables Beretta to use interval trees to store constraints on numeric attributes, thereby yielding an efficient matching algorithm [7]. Moreover, subsumption after subscription splitting also helps avoid the problem of subscription summaries in CPS systems becoming unmanageable [7]. To enable Beretta to sustain its performance even under high churn, i.e. during addition, deletion and modification of subscriptions, all subscriptions in Beretta are parameterized. Our EventJava compiler introduces variables into all the normalized subscriptions of Beretta, thereby handling all subscription changes as subscription updates.

Future Research Directions

In the long term, I expect my research to focus on the broader theme of "Effectively Engineering Large-Scale Distributed Systems". In the medium-term, some of the topics I am excited about include:

Programming Languages/Frameworks for Elasticity. One of my research objectives in the area of programmable elasticity is to design an elastic variant of the Java programming language – **ElasticJava**. The motivation behind ElasticJava is to design a generic programming language for elastic distributed programs, which is easy to use, familiar to a large community of programmers and supports a variety of programming idioms. I plan to realize ElasticJava in a modular way, first by extending my work on Elastic Remote Methods [3] to other programming styles – incorporating elasticity into distributed fork-join, actors, event-based programming and Java futures, and integrating the resulting frameworks. I expect this research to provide valuable insights into the types of applications for which explicit elasticity is most useful.

Migration of Legacy Distributed Applications to the Cloud: When legacy *distributed* applications designed to be executed efficiently on dedicated data centers are migrated to cloud computing environments (e.g., IaaS clouds like EC2), the application will not be able to exploit the elasticity and auto scalability features of the cloud without significant refactoring. I am interested in programming language techniques and tools that enable legacy enterprise applications to be efficiently migrated to the cloud, i.e., that automate the migration process as much as possible. Specifically, I am interested in (1) static analyses that use programmer input to identify application components which have to be instantiated during auto-scaling, and (2) program transformations (both source-code transformations and binary instrumentation) that create new instances of these components at runtime by requesting resources (e.g, VM instances) from the cloud management system. This is an important research problem because technology research firms have predicted that 80% of all server x86 workloads will execute on VMs by 2018. (Gartner Inc. Oct 2012)

Distributed Algorithms for NVRAM-based Datacenters: Non-volatile random-access memory (NVRAM) allows creation of persistent data, i.e. data that lives beyond the lifetime of the creating process, and beyond the next hardware or software failure. With block devices such as disks, persistence can be achieved; but moving data to the persistent store is slow and requires translation between the application object format and the persistent file format. SSDs improve matters significantly, and have been used as caches for hard disks to improve performance. But, access times remain orders of magnitude slower than primary memory. One of my goals is to expand the benefits of single-node NVRAM to distributed systems in a datacenter with many compute nodes having NVRAM. In this context, I am interested in developing distributed shared memory frameworks (1) exporting a global persistent address space that can be shared among data center applications, (2) allowing unordered, ordered, and transactional access to locations in a persistent region, (3) incorporating efficient distributed algorithms that enable datacenter infrastructure applications like key-value stores, file systems, name services and coordination services like lock managers to achieve strong guarantees (atomicity, consistency, isolation and durability for example) with minimal sacrifices to their performance.

Event Correlation in Large Scale Graph Stores: With the advent of large graphs in social, road-traffic and biological networks, several graph storage systems have been developed e.g., Trinity, GraphLab and our own Concerto [11]. Such systems optimize graph storage and traversal e.g., by using a pointer-chasing layout which is not readily suitable for large-scale event stream processing. On the other hand, traditional overlay networks in stream processing and publish/subscribe systems are not suitable for graph traversals like depth-first-search because they do not attempt to co-locate neighboring graph nodes. Both graph traversals and event processing are important in e.g., social networks – graph traversals for analytics workloads and event processing for handling the hundreds of millions of updates to social network data every day. Hence, I am interested in a reliable, geo-replicated, storage framework for large-graphs which optimizes both graph traversals and event processing. I am also very interested in exploring the challenges in implementing such storage systems on non-volatile RAM, and whether such implementations can drastically reduce their latency and improve their throughput.

Decentralized Online Advertising: I am also excited about applying event-based design to new application domains, especially online advertising. In particular, I would like to investigate the design and implementation of decentralized advertisement (ad) exchanges, that leverage the geo-distribution advantage of cloud-computing platforms to enable advertisers bid in (soft) real time for the placement of ads. Applying publish/subscribe systems to design ad exchanges has several important challenges, including but not limited to (1) augmenting CPS systems with algorithms to collect reliable metrics about the distribution of subscribers, (2) accurately measuring the number of users that get an advertisement, and (3) satisfying the bounds placed by advertisers on the number of times an advertisement can be delivered. Furthermore, the approximate nature of subscriptions in ad exchanges – an advertiser can specify that a user should get an ad when he/she matches a certain threshold (e.g., 80% of the constraints set by the advertiser) – necessitates the development of scalable and efficient algorithms that perform "approximate" matching of advertisements to users. Implementing an ad exchange in the cloud introduces its own set of challenges, including but not limited to the assignment of advertisers and users to servers in the ad network and distributed storage of video ads.

Beyond the problems and topics outlined above, I am always looking to move into new areas and collaborate on other challenging problems.

References

[1] Han Chen, Alexei Karve, Minkyong Kim, Andrzej Kochut, Jayaram K. R., Hui Lei, Zhiming Shen, and Zhe Zhang. Virtual machine image access deduplication. *Patent application, filed in 2012*, 2012.

[2] Patrick Eugster and K. R. Jayaram. Eventjava: An extension of java for event correlation. In *Proceedings of the 23rd European Conference on ECOOP 2009 — Object-Oriented Programming*, Genoa, pages 570–594, Berlin, Heidelberg, 2009. Springer-Verlag.

[3] K. R. Jayaram. Elastic remote methods. In *Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware*, MIDDLEWARE '13, 2013.

[4] K. R Jayaram and Patrick Eugster. Scalable efficient composite event detection. In Dave Clarke and Gul Agha, editors, *Proceedings of the International Conference on Coordination Models and Languages (COORDINATION)*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010.

[5] K. R. Jayaram and Patrick Eugster. Program analysis for event-based distributed systems. In *Proceedings of the 5th ACM international conference on Distributed event-based system*, DEBS '11, pages 113–124, New York, NY, USA, 2011. ACM.

[6] K. R. Jayaram and Patrick Eugster. Program Analysis for Event-based Distributed Systems. In *Proceedings of the 5th ACM International Conference on Distributed Event-based Systems*, DEBS '11, pages 113–124, New York, NY, USA, 2011. ACM.

[7] K. R. Jayaram and Patrick Eugster. Split and Subsume: Subscription Normalization for Effective Content-Based Messaging. In *Proceedings of the 2011 31st International Conference on Distributed Computing Systems*, ICDCS '11, pages 824–835, Washington, DC, USA, 2011. IEEE Computer Society.

[8] K. R. Jayaram, Patrick Eugster, and Chamikara Jayalath. Parametric content-based publish/subscribe. *ACM Trans. Comput. Syst.*, 31(2):4:1–4:52, May 2013.

[9] K. R. Jayaram, Chamikara Jayalath, and Patrick Eugster. Parametric subscriptions for content-based publish/subscribe networks. In *Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware*, MIDDLEWARE '10, pages 128–147, Berlin, Heidelberg, 2010. Springer-Verlag.

[10] K. R. Jayaram, Chunyi Peng, Zhe Zhang, Minkyong Kim, Han Chen, and Hui Lei. An Empirical Analysis of Similarity in Virtual Machine Images. In *Proceedings of the 12th ACM/IFIP/USENIX International Middleware Conference*, MIDDLEWARE '11, 2011.

[11] Michael M. Lee, Indrajit Roy, Alvin AuYoung, Vanish Talwar, K. R. Jayaram, and Yuanyan Zhou. Views and transactional storage for large graphs. In *Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware*, MIDDLEWARE '13, 2013.