Multiplicity Computing: A Vision of Software Engineering for Next-Generation Computing Platform Applications

Cristian Cadar, Peter Pietzuch, and Alexander L. Wolf

Department of Computing
Imperial College London
London, UK

ABSTRACT

New technologies have recently emerged to challenge the very nature of computing: multicore processors, virtualized operating systems and networks, and data-center clouds. One can view these technologies as forming levels within a new, global computing platform. We aim to open a new area of research, called *multiplicity computing*, that takes a radically different approach to the engineering of applications for this platform. Unlike other efforts, which are largely focused on innovations within specific levels, multiplicity computing embraces the platform as a virtually unlimited space of essentially redundant resources. This space is formed as a whole from the cross product of resources available at each level in the platform, offering a "multiplicity" of end-to-end resources. We seek to discover fundamentally new ways of exploiting the combinatorial multiplicity of computational, communication, and storage resources to obtain scalable applications exhibiting improved quality, dependability, and security that are both predictable and measurable.

Categories and Subject Descriptors

C.2.0 [**Computer-Communications Networks**]: General; D.2.0 [**Software Engineering**]: General; D.4.0 [**Operating Systems**]: General

General Terms

Design, Experimentation, Measurement, Performance, Reliability, Security

1. WHY MULTIPLICITY COMPUTING?

The first decade of the 21st century has seen the emergence of several new computing technologies: multicore processors, virtualized operating systems and networks, and data-center clouds. Simplifying a bit, one can view these technologies as forming levels within a new, global computing platform for application software systems, the "Internet computer". Application developers are looking to this platform to provide the computational, communication, and storage resources they need in order to offer services to unlimited numbers of users around the globe in a cost-effective and efficient manner [20]. The dream of computing as a utility much like the electric power grid would seem to be approaching reality.

Many current applications can take immediate benefit from the new computing platform, and this simple fact explains the rapidity with which the various platform technologies have gained such wide acceptance. With their early success, significant interest is building in drastically expanding the capabilities of the platform technologies. Indeed, the growth trends predicted for them are staggering, with estimates of orders of magnitude more computing resources becoming available within the next decade [2, 13, 16].

It is a critical time, therefore, to ask the following difficult questions: *How can applications take advantage of this exponential growth in power? Is it enough to continue using the current methods for developing applications? What new approaches to application development are made possible in the context of the new computing platform?*

Any given application in existence today, bound by the limits of its structure and semantics, is unlikely to saturate the resources provided. In fact, there are already techniques being developed to avoid the use of some portion of the cores in a processor simply because they consume energy without necessarily providing increased performance [17].

What we see is an approaching plateau, where applications will no longer benefit from the growing power of the computing platform unless those applications undergo a fundamental reconceptualization. This effect will be mirrored in the software engineering techniques intended to develop and support these applications; the utility and viability of current techniques are threatened by the sheer scale to which the platform is expected to evolve.

The currently popular idea is to seek greater parallelism within the various levels of the platform. This has led to a renewed interest in parallelizing compilers and functional languages, and to the development of large-scale, coarse-grained parallel data-processing systems, such as MapReduce [8], Hadoop,[1] and Dryad [14]. These efforts are producing important results, and continued work on them is well justified. But simply concentrating on level-specific improvements does not constitute a comprehensive strategy for fully exploiting tomorrow's vastly richer computing environ-

[1] `http://hadoop.apache.org/`

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
FoSER 2010, November 7–8, 2010, Santa Fe, New Mexico, USA.
Copyright 2010 ACM 978-1-4503-0427-6/10/11 ...$10.00.

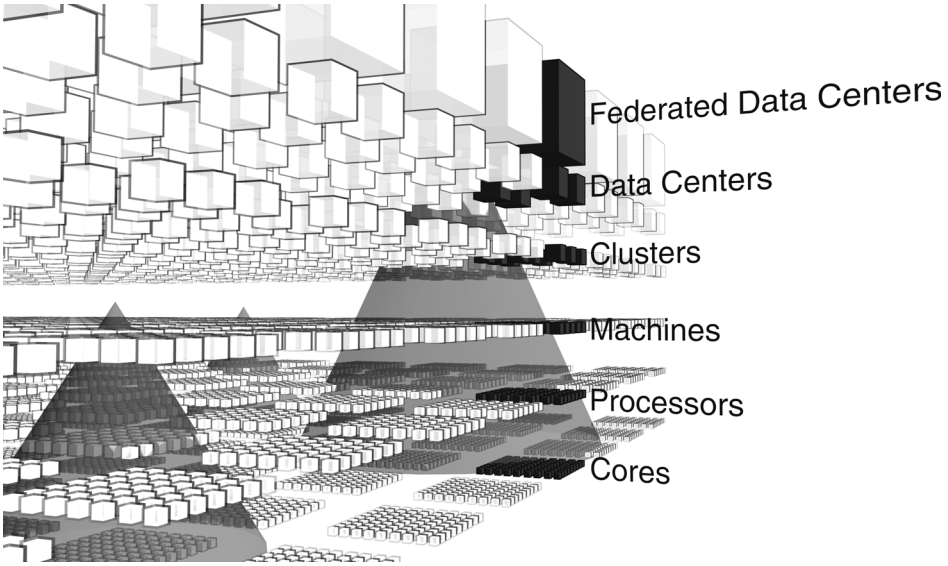

Figure 1: Multiplicity of Computational Resources used by Next-Generation Applications

ment. We will get techniques that are merely effective within a given level, but not necessarily effective when placed in concert with techniques employed at other levels.

We envision a new area of research that takes a radically different approach to investigating the opportunities presented by the next-generation computing platform. We refer to this new area as *multiplicity computing*, embracing the platform as a virtually unlimited space of essentially redundant resources. This space is formed as a whole from the cross product of resources available at each level in the platform, offering a "multiplicity" of end-to-end resources. The concept is illustrated in Figure 1, which shows the various layers of platform technologies underlying multiplicity computing. In particular, the figure shows the coming multiplicity of *computational resources*. Not shown, but similarly abundant, will be *communication* and *storage resources*; any given application will require resources of all three kinds. Applications are depicted as shaded cones that have allocated to them, within the cones, some subset of the available resources. Notice that applications can be deployed at any level, such as in a data center across multiple machines or simply within an individual machine. And, although not illustrated in the figure, the allocated subsets of different applications may in fact overlap. Of course, virtualization and multi-tenancy technologies also play vital roles, abstracting the resources in such a way that they can be shared and reallocated without disturbing the applications using them.

A concept related to, but distinct from, multiplicity computing is *multi-scale computing*, which seeks an integration of computing at different scales, from the fine-grained processing of environmental data in sensor nodes to context-driven applications running on mobile devices to bulk processing of data on supercomputers or in data centers. For example, the University of California's Multiscale Systems Center[2] is looking at this problem from the perspective of "distributed sense and control". The research questions being asked are therefore quite different.

[2] `http://www.musyc.org/`

2. A SIMPLE SCENARIO

The following scenario gives a concrete example of the challenges and potential of multiplicity computing. Consider a global social-networking service that provides an infrastructure for hundreds of millions of users world wide to communicate, collaborate, and interact. As a platform for third-party applications, it enables an ecosystem of new social, business, and entertainment applications to be sold through a digital marketplace on the site.

The immense resource requirements of such an infrastructure are satisfied by hundreds of cloud data centers located around the world. The infrastructure operator extensively uses virtualization for resource isolation and must manage this global infrastructure efficiently. This involves balancing the requirements of individual applications in terms of performance, availability, and reliability with its own goals as an operator regarding efficiency and utilization of the platform as a whole. For example, the operator's desire to reduce energy costs may mean that underutilized racks of thousands of computers should be switched off at times and components reallocated to other racks perhaps in other data centers, thereby accepting a small reduction in application performance for improved energy efficiency.

Software engineers must be given the right abstractions to design, build, and deploy applications in this complex environment. They must understand what computational, communication, and storage resources they require when deploying, for example, a new graphics-intensive, multi-player, virtual-reality game. The game may require redundant resources in multiple data centers to guarantee its availability, but must also dynamically acquire additional resources to handle flash crowds in parts of the virtual game world. Guaranteeing the availability of such an application does not only mean guarding against resource failures, but also protecting against software bugs and security attacks. For instance, when rolling out updates of applications on this infrastructure, there is a risk that new software flaws may lead to catastrophic application failures.

3. OPPORTUNITIES

Multiplicity computing offers new ways to deal with the issues highlighted in the scenario described above. To give a flavor of our vision, we outline several new ideas that could be explored through multiplicity computing.

Resiliency through diversity. Rather than thinking of an application, even a highly distributed application consisting of many components, as a single entity to be deployed and executed on the computing platform, what if we could deploy and execute hundreds or thousands of instances of the application simultaneously? If those instances were actually slight variants of each other, perhaps generated automatically (e.g., through genetic programming [10]), then there is a statistical likelihood that their performance and/or quality properties differ in some interesting way, such as freedom from some bug or security vulnerability.

Staged deployment. The current processes for upgrading an application, especially a "non-stop" application, typically require a complex and coordinated effort among developers, operators, and users, and usually involve some amount of service interruption. Many such processes have proven disastrous (e.g., the recent incidents involving Skype and Google's GMail), either because of configuration problems or simply low quality in the new versions. What if multiple upgrade candidates could be brought into service and allowed to execute in parallel with old versions? Users could then select the best candidate according to their needs, and incrementally migrate to that version as confidence in it grew.

Intelligent parameter-space exploration. Large-scale distributed applications exhibit emergent behaviors that are typically not amenable to formal reasoning. This is due to the extremely large, highly dimensional, and somewhat uncontrollable space of parameters that affect and determine the execution of an application. These parameters include the basic functional parameters of the application (i.e., "inputs") as well as those that describe the platform (e.g., network latency and machine failure rates). What if we could explore that space experimentally by choosing a series of trial runs relative to some hypothesis (e.g., the relationship among some set of parameter values), and simultaneously deploy and execute hundreds or thousands of trials to prove or disprove that hypothesis? This would allow a substantial increase in the predictability of application behavior.

Cross-boundary resource optimization. The increasing abundance of computing resources will almost certainly be accompanied by a sharp drop in the unit cost of computing. Over time, however, applications tend to use more units rather than the same number of units at lower cost (typically because their functionality increases, disciplined use of resources decreases, or both). In the end, operating costs will remain a critical issue. For example, an application may give up locally available cores for a larger number of cores in a remote data center, accepting the incurred increase in communication latency. What if we could freely optimize cost both horizontally and vertically across the platform, taking into account the different properties of each platform level when allocating resources to applications?

Notice that some of these ideas have to do with the design of applications, some with the validation of those designs, and some with the operation of applications. Thus, multiplicity computing would have an impact on nearly every aspect of how we go about the engineering of applications.

4. SOME RESEARCH THEMES

Our vision for multiplicity computing begins a process of thinking that should be expected to broadly influence a number of core computing disciplines, specifically software engineering, distributed systems, operating systems, and networking. It should also influence the development of software in a broad range of application areas. We now sketch just a few of the many research themes that could be investigated under the rubric of multiplicity computing.

Multi-level platform management. Current solutions for resource management only focus on specific platform levels, such as the scheduling of processes on cores in Barrelfish [3] or the migration of virtual machines [23]. The lack of coordinated decision making at different levels hides the trade off between horizontal and vertical scaling [20] and leads to fragmented resource allocation. It also makes it difficult to guarantee high-level policies for efficiency, elasticity, and availability. An example of a policy that is currently hard to implement is to power down automatically an entire cluster within a data center when the application workload can be handled by the remaining clusters. Without support for such high-level management policies, multiplicity computing risks wasting physical resources and thereby provide poor service to applications.

To address this problem, we argue that resource management should take a holistic view of resources across multiple platform levels. This enables resource management decisions to trade off resources at different levels, as suggested by our earlier example of trading the location of cores against communication latency. By providing a unified model of resources in multiplicity computing, we can enable better management approaches that guarantee long-term goals, such as those relating to power consumption.

Multi-level platform management opens several new research challenges: (1) scalable and dynamic resource discovery and monitoring [18]; (2) decentralized algorithms for holistic resource allocation that guarantee global properties; (3) techniques for cross-level optimization of resource management; and (4) approaches for specification and enforcement of high-level management policies [9].

Speculative execution. The next-generation platform creates new opportunities for exploiting vast computing resources. Current approaches have mainly focused on improvements to application performance. Unfortunately, this works well only for applications with a great amount of inherent parallelism, that is, those that can be divided into multiple computations that can be solved concurrently.

Multiplicity computing could allow us to broaden our attention from simple performance to additionally address concerns of reliability, availability, and security. This would allow us to improve a much larger class of applications, including those with limited or no inherent parallelism. The key observation is that we can use the extra available computing power to create different variants of the same application, execute those variants in parallel, and dynamically choose the best one available for the task at hand. We refer to this as *speculative execution*. Examples of where speculative execution should have immediate benefit include: avoiding erroneous order dependencies in distributed message queues by executing variants that process the queues in different orders; avoiding vulnerable portions of memory by executing variants having different memory layouts [4];

avoiding pathological behaviors in heuristic algorithms by executing variants having different non-deterministic properties [12]; and avoiding regression in bug fixing by executing variants with different patch sets, which could be generated by genetic programming [10] or symbolic execution [5]. This is a much broader set of goals than that of the prior work of Cledat et al. [7] or Forrest et al. [10].

The process of applying speculative execution would consist of three main tasks. First we generate large numbers of multiple variants of an application whose quality we wish to improve, thus forming what we term an "application family". These variants are then allocated physical resources according to certain semantic constraints and the resources available. Finally, the execution of the application family is coordinated via an orchestration strategy that will involve complex state synchronization across the variants. Thus, three research challenges of speculative execution are: (1) generating large application families automatically; (2) managing the resources allocated to an application family; and (3) coordinating the execution of an application family.

Application design principles. Multiplicity computing has enormous potential to enable new classes of applications at an unprecedented scale. But it also bears the risk that application developers will be overwhelmed by the complexity of designing and implementing applications. The sheer number of resources require new software architectures to take advantage of them. Applications must be tailored towards this environment by building them from a large number of independent software components. These components must communicate, coordinate, and interact with each other efficiently. If developers are not given the right abstractions and programming primitives to achieve this, they will be unable to leverage the benefits of multiplicity computing.

Based on existing application success stories in cloud data centers, we can infer simple design principles for suitable software architectures. Parallel computation frameworks, such as MapReduce, Hadoop, and Dryad, require algorithms to be expressed in simple functional terms, making them extremely parallelizable. Data storage in wide-area distributed file systems, as in GFS [11] and WheelFS [21], relies heavily on redundant copies of data to provide fault tolerance, and on eventual consistency for scalability and availability. Group communication mechanisms using publish/subscribe and epidemic gossip-style dissemination have been shown to scale to a large number of components [6, 15, 19].

We argue that it is necessary to provide *design principles for multiplicity computing applications*. These principles will enable application developers to adopt architectures that can take advantage of parallelism, elasticity, and isolation. In addition, new primitives and abstractions for communication, orchestration, and storage are needed. Thus, we can see five research challenges related to application design principles: (1) software architectures for multiplicity computing; (2) group communication mechanisms for vast numbers of components; (3) primitives for generating large application families; (4) abstractions for efficient orchestration and synchronization of components; and (5) support for scalable persistence of static and dynamic data.

Guided experimentation. Experimental exploration of an application's parameter space—that is, the setting of independent variables leading to measurable effects on dependent variables of interest—is today largely a manual process: relying on experience and intuition, engineers select a sequence of sample points to test some hypothesis about the relationship among the variables. Correspondingly, the state of the art in distributed-system experimentation is based on the general notion of a *network testbed* whose support tools (e.g., Plush [1] for PlanetLab or our own general-purpose tool Weevil [22]) only provide what amount to a sophisticated deployment and execution environment for a distributed system and its experimental workload. The tasks of experimental design and, particularly, parameter-space exploration are left to the engineer.

Multiplicity computing offers a fresh perspective: the same abundance of resources available for the normal operation of applications can be exploited to carry out a richer experimentation process. Moreover, multiplicity computing's multi-level platform management can provide a sophisticated means to control the allocation of experimental resources, while its speculative execution can provide novel ways of generating and orchestrating experimental trials. But with all this richness comes added complexity and, therefore, a need to provide substantial help to the engineer as they go about designing experiments in this new context.

We see experimentation as *guided exploration*: performing experiments by providing guidance to an integrated framework of automation tools for constructing experimental trials from a specification of experimental hypotheses and parameter-space dimensions, executing trials and feeding their results back into the trial construction process, and discovering the interplay of parameters and deriving new experimental hypotheses. While prior research has looked at automating parameter-space exploration for limited kinds of parameters, for limited kinds of behaviors, for centralized systems, or for degenerate, server-centered distributed systems, never before has the problem been attempted for large-scale distributed systems in their full richness and complexity. Thus, several fundamental advances will be required to support our broader vision, summarized as the following set of research challenges: (1) models and modeling formalisms capable of capturing experimental hypotheses, the elements and boundaries of a highly dimensional parameter space, key performance indicators, and an overall cost model for the experimentation process; (2) parameter-space sampling techniques based on sound theoretical foundations and empirically justified heuristics, and suited to the wide variety of parameters in the distributed-system domain; and (3) hypothesis inference techniques effective at discovering useful new experimental questions and parameter relationships.

5. CONCLUSION

We are witnessing a step change in the basic computing platform, characterized fundamentally by the abundance of computational, communication, and storage resources on offer to any and all application systems. Multiplicity computing represents a radical rethink of how those resources can be leveraged by software engineers, not merely to increase performance or decrease cost, but to obtain scalable applications exhibiting improved quality, dependability, and security that are both predictable and measurable.

Acknowledgements. We thank Stuart Hall, Julian Hodgson, and Andrew Ruhemann of Passion Pictures (http://www.passion-pictures.com) for their imaginative depiction of resource multiplicity in Figure 1.

6. REFERENCES

[1] J. Albrecht, C. Tuttle, A. Snoeren, and A. Vahdat. Distributed application management using Plush. In *14th IEEE International Symposium on High Performance Distributed Computing*, pages 281–282, Research Triangle Park, North Carolina, July 2005.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A Berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, University of California, Berkeley, Feb. 2009.

[3] A. Baumann, P. Barham, P.-E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhania. The multikernel: A new OS architecture for scalable multicore systems. In *ACM SIGOPS 22nd Symposium on Operating Systems Principles*, pages 29–44, New York, New York, USA, 2009. ACM.

[4] E. Bhatkar, D. C. Duvarney, and R. Sekar. Address obfuscation: An efficient approach to combat a broad range of memory error exploits. In *12th USENIX Security Symposium*, pages 105–120, 2003.

[5] C. Cadar, V. Ganesh, P. Pawlowski, D. Dill, and D. Engler. EXE: Automatically generating inputs of death. *ACM Transactions on Information and System Security*, 12(2):1–38, 2008.

[6] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[7] R. Cledat, T. Kumar, J. Sreeram, and S. Pande. Opportunistic computing: A new paradigm for scalable realism on many-cores. In *1st USENIX Workshop on Hot Topics in Parallelism*, Berkeley, California, Mar. 2009. USENIX Association.

[8] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *6th Symposium on Operating Systems Design and Implementation*, Berkeley, California, USA, 2004. USENIX Association.

[9] D. M. Eyers, R. Routray, R. Zhang, D. Willcocks, and P. Pietzuch. Towards a middleware for configuring large-scale storage infrastructures. In *7th International Workshop on Middleware for Grids, Clouds and e-Science*, Urbana Champaign, Illinois, USA, Nov. 2009. ACM/IFIP/USENIX.

[10] S. Forrest, W. Weimer, T. Nguyen, and C. L. Goues. A genetic programming approach to automated software repair. In *Genetic and Evolutionary Computing Conference*, pages 947–954, 2009.

[11] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. *SIGOPS Operating Systems Review*, 37(5):29–43, 2003.

[12] Y. Hamadi, S. Jabbour, and L. Sais. ManySAT: A parallel SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 6:245–262, 2009.

[13] J. Held, G. Bautista, and S. Koehl. From a few cores to many: A tera-scale computing research overview. White Paper, Intel Corporation, 2006.

[14] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. *SIGOPS Operating Systems Review*, 41(3):59–72, 2007.

[15] A.-M. Kermarrec and M. van Steen. Gossiping in distributed systems. *SIGOPS Operating Systems Review*, 41(5):2–7, 2007.

[16] K. L. Kroeker. The evolution of virtualization. *Communications of the ACM*, 52(3):18–20, Mar. 2009.

[17] J. Leverich, M. Monchiero, V. Talwar, P. Ranganathan, and C. Kozyrakis. Power management of datacenter workloads using per-core power gating. Technical Report HPL-2009-326, HP Laboratories, Sept. 2009.

[18] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Distributed resource discovery on Planetlab with SWORD. In *1st Workshop on Real, Large Distributed Systems*, San Francisco, California, Dec. 2004.

[19] I. Rose, R. Murty, P. Pietzuch, J. Ledlie, M. Roussopoulos, and M. Welsh. Cobra: Content-based filtering and aggregation of blogs and RSS feeds. In *4th USENIX Symposium on Networked Systems Design and Implementation*, Cambridge, Massachusetts, USA, Apr. 2007.

[20] L. Schubert, K. Jefferey, and B. Neidecker-Lutz. The future of cloud computing. Expert Group Report, European Commission, 2010.

[21] J. Stribling, Y. Sovran, I. Zhang, X. Pretzer, J. Li, M. F. Kaashoek, and R. Morris. Flexible, wide-area storage for distributed systems with WheelFS. In *6th USENIX Symposium on Networked Systems Design and Implementation*, Boston, Massachusetts, Apr. 2009.

[22] Y. Wang, A. Carzaniga, and A. L. Wolf. Four enhancements to automated distributed system experimentation methods. In *13th International Conference on Software Engineering*, pages 491–500, Leipzig, Germany, May 2008.

[23] T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif. Black-box and gray-box strategies for virtual machine migration. In *4th Symposium on Networked Systems Design and Implementation*. USENIX Association, 2007.