Does Disaggregrated Compute Require a New Programming Paradigm?

Jeremy Singer Colin Perkins Herry Herry

February 27, 2018

We are now in the era of *disaggregated* compute [5], where systems are composed of many small, heterogeneous, highly distributed nodes. The growing *edge* [8] and *fog* [2] compute trends are special cases of disaggregation, for which, respectively, compute-intensive operations occur at network end-points or in near-edge servers. Example scenarios include applications where data acquired at the network edge must be processed locally, perhaps for privacy reasons, or to ensure low-latency response times.

Our Federated Raspberry Pi micro-Infrastructure Testbed (FRμIT) project is building a system composed of federated set of micro-clusters of single board computers. Each micro-cluster has a set of Raspberry Pi nodes connected over a LAN, with an internet uplink. An explicit aim for the FRμIT system is to be as decentralized as possible, i.e. services should operate using peer-to-peer (p2p) architectural design patterns [9]. Our current framework is semi-distributed but we are moving to a fully distributed solution.

We aim to build a toolchain to enable the development and deployment of composite, distributed applications that run across the whole federated system, or possibly across an arbitrary subset of the nodes. Current state of the art is to deploy single-node executables inside containers, based on Docker or similar frameworks. A whole system is configured using a higher level orchestration framework, perhaps Kubernetes. The glaring problem with this approach is the *requirement to use multiple languages*. The low-level, single-node programs are written in conventional programming languages. The containers have associated metadata, which can be queried and updated using a domain-specific language. Wide-area configuration and deployment is expressed in another domain-specific language, typically a dialect of YAML, and applied with a sequence of commands.

Our research investigates whether it is possible to adopt a unified approach to distributed system development on FRμIT style platforms. The key question we intend to address is, *are any existing programming languages appropriate for this use case?*

Languages based on theoretical process calculi, including Occam [7] (from CSP) and Pict [6] (from π-calculus), might be appropriate. These incorporate the notions of parallelism, message passing, and can be extended to model process mobility—for instance, Nomadic Pict [10] is described as a 'wide area programming language' which sounds like our use case. The key issue is that we cannot expect all nodes to be reachable at all times. Nodes may drop off temporarily (perhaps due to power or network problems) or permanently (perhaps due to hardware failure).

Modern concurrent systems incorporate runtime notions of failure-tolerance, such as the actor-based Erlang system [1] and the task-parallel Hadoop/MapReduce framework [4]. We note that other programming languages build on these archetypes, such as Elixir (for Erlang) and Spark (for Hadoop).

More specialized languages include Ambient Talk [3] which targets sensor network deployments. It is based on p2p mobile architecture and has explicit network failure support.

We expect that a combination of these properties will be required for our new scenario. Effectively, we will need a toolchain that meets the following set of high-level requirements:

1. a unified language to express computation (data processing), communication (message passing) and configuration (task deployment).
2. built-in support for delay tolerance and failure tolerance
3. coverage of heterogeneous device nodes across the system
4. fundamental support for a fully decentralized architecture
5. expression of resource constraints and capabilities associated with each node, with associated scheduling support
6. facilities for compositional development of large applications

References

[1] J. Armstrong. A history of erlang. In *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages*, pages 6–1–6–26, 2007.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, pages 13–16, 2012.

[3] T. V. Cutsem, E. G. Boix, C. Scholliers, A. L. Carreton, D. Harnie, K. Pinte, and W. D. Meuter. Ambienttalk: programming responsive mobile peer-to-peer applications with actors. *Computer Languages, Systems & Structures*, 40(3):112–136, 2014.

[4] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[5] H. Meyer, J. C. Sancho, J. V. Quiroga, F. Zyulkyarov, D. Roca, and M. Nemirovsky. Disaggregated computing. an evaluation of current trends for datacentres. *Procedia Computer Science*, 108:685–694, 2017.

[6] B. C. Pierce and D. N. Turner. Pict: a programming language based on the pi-calculus. In *Proof, language, and interaction*, pages 455–494, 2000.

[7] A. W. Roscoe and C. A. R. Hoare. The laws of occam programming. *Theoretical Computer Science*, 60(2):177–229, 1988.

[8] M. Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, Jan 2017.

[9] J. Singer, H. Herry, P. J. Basford, W. Hajji, C. S. Perkins, F. P. Tso, D. Pezaros, R. D. Mullins, E. Yoneki, S. J. Cox, and S. J. Johnston. Next generation single board clusters (demo), 2018. to appear in Proceedings of the 2018 IEEE/IFIP Network Operations and Management Symposium.

[10] P. T. Wojciechowski and P. Sewell. Nomadic pict: Language and infrastructure design for mobile agents. *IEEE Concurrency*, 8(2):42–52, 2000.