Position paper: Toward an holistic approach of Systems of Systems

Simon Bouget

Position paper: Toward an holistic approach of Systems of Systems

Simon Bouget
IRISA – Université de Rennes 1
simon.bouget@irisa.fr

ABSTRACT

Large scale distributed systems have become ubiquitous, from on-line social networks to the Internet-of-things. To meet rising expectations (scalability, robustness, flexibility,...) these systems increasingly espouse complex distributed architectures, that are hard to design, deploy and maintain. To grasp this complexity, developers should be allowed to assemble large distributed systems from smaller parts using a seamless, high-level programming paradigm. We present such an assembly-based programming framework, enabling developers to easily define and realize complex distributed topologies as a construction of simpler blocks (e.g. rings, grids). It does so by harnessing the power of self-organizing overlays, that is made accessible to developers through a high-level Domain Specific Language and self-stabilizing runtime. Our evaluation further shows that our approach is generic, expressive, low-overhead and robust.

1. INTRODUCTION

Modern distributed applications are becoming increasingly large and complex. They often bring together independently developed sub-systems (e.g. for storage, batch processing, streaming, application logic, logging, caching) into large, geo-distributed and heterogeneous architectures [10]. Combining, configuring, and deploying these architectures is a difficult and multifaceted task: individual services have their own requirements, configuration spaces, programming models, distribution logic, which must be carefully tuned to insure the overall performance, resilience, and evolvability of the resulting system.

This integration effort remains today largely an ad-hoc activity, that is either manual or uses tool-specific scripting capabilities. This low-level approach unfortunately scales poorly in the face of the increasingly complex deployment requirements and topologies of the involved services [13, 9, 16, 20].

The lack of a principled and systematic programming model that is able to consider existing distributed systems as composable first class entities imposes a high toll on developers. In order to write and maintain the low level glue code or configuration files required to realize these topologies, they must (i) have a *deep understanding* of the involved distributed services, their specific semantics, and individual programming model ; (ii) cater for the *unavoidable volatility* of the workloads and of the cloud infrastructures in which these services typically operate; and (iii) allow for a *continuous integration* process in which a deployed system is modified on the fly.

To solve this situation, we argue that practitioners should be allowed to programmatically manipulate distributed systems as *first class entities* [3], from which whole distributed systems can be *incrementally assembled*.

We also argue that the mapping of systems to individual nodes should remain as much as possible transparent to developers. In particular developers should not have to worry about nodes failing, leaving or joining the system (a common occurrence in public clouds for instance), or about the intricacies of scaling operations.

As a first step towards this ambitious goal, we propose an assembly-based programming framework for the implementation of complex distributed topologies. It provides developers with a high level component-based programming model [8, 5], and exploits self-organizing overlays [23, 2, 11] to map at runtime a developer's high-level description of a complex distributed topology onto a concrete infrastructure. It relies on the scalability, resilience, and adaptability of self-organizing overlays to maintain a developer's target topology in the face of failures, scaling and dynamic adaptations.

2. STATE OF THE ART

Easing the development of complex distributed systems has been a long-running and recurrent objective of middleware research. Most of these efforts have however focused on the local behavior of individual nodes (e.g. with protocol kernels [21, 15], or component frameworks [8, 5, 19]), rather than on the programmatic means to describe a system's global structure and behavior. As a result, most of these programming frameworks offer little or no support for the flexible integration of individual systems into a larger whole.

2.1 Component-based programming

Component-based software engineering (CBSE) promotes *development by assembly*. It allows developers to construct complex systems by assembling pre-existing *components*, i.e. modular reusable blocks that explicitly exposes their interfaces—

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

both in terms of requirements and of features provided. Components provide *separation of concerns* and *modularity*, and facilitate re-use and continuous integration. A large number of component technologies have been successfully applied to distributed systems over the years, both in industry (e.g. *Enterprise Java Beans* (EJB), the *Service Component Architecture* (SCA), the *CORBA Component Model* (CCM), .Net, and the *OSGi Remote Services* Specification) and academia [5, 8].

These solutions, however, view components as software artifacts living *within* nodes, and focus therefore on the workings of individual nodes rather than on a system's global behavior. By contrast, we propose to inverse this view, and consider components as *distributed entities* enforcing a given internal structure (a star, a tree, a ring) which developers can assemble programmatically to realize more complex topologies. Individual nodes now live within components, and become transparent to developers, who only perceive system-level entities they can instantiate and connect to form larger wholes.

2.2 Self-organizing overlays

To realize this vision, we propose to exploit self-organizing overlays [11, 23, 2], a family of decentralized protocols that are able to autonomously organize a large number of nodes into a predefined topology —from a random network [12] to a ring or torus [22, 11] to an hypercube— by exploiting epidemic (or gossip) interactions to progressively organize nodes. Self-organizing overlays are self-healing, and can with appropriate extension, conserve their overall shape even in the face of catastrophic failures [4]. These topologies can be used to support the many P2P- and cloud-based applications that have been proposed for over a decade now, such as VoIP (e.g. Skype), streaming [24], pub-sub [6], and storage [18, 9]. In particular, the scalability and robustness of these solutions have made them particularly well adapted to large scale self-organizing systems such as decentralized social networks [14, 2], news recommendation engines [1], and peer-to-peer storage systems [7].

However, more and more applications require much more complex topologies [9, 13] that can be hard to obtain *via* the traditional protocols. Typically, self-organizing overlays such as T-Man [11] or Vicinity [23] are unfortunately *monolithic* in the sense that they rely on a single user-defined *distance function* to connect nodes into a target structure, e.g. nodes try to reach and connect to the "closest" nodes in their ID space. Simple topologies such as ring or torus are easy to realize in this model, but more complex combinations, such as a star of cliques, are more problematic. This model does not lend itself naturally to development by assembly, mentioned in the previous section: self-organizing overlays, in their basic form, have no notion of composition or connection to other overlays.

As a conclusion, by merging techniques from the two domains, our approach goes beyond both of them: (i) On one hand, beyond traditional component-based frameworks for distributed systems in that it considers *components* as *collective distributed entities* enforcing a given internal structure (a star, a tree, a ring) which developers can assemble programmatically to realize more complex topologies. (ii) On the other hand beyond existing self-organizing overlays by supporting the description of a target topology as a *composition of more elementary shapes*, breaking away from the monolithic design of typical self-organizing overlay protocols.

This enables a programmer to create, deploy and maintain easily the more complex topologies that are needed to support today's sophisticated applications, such as distributed NoSQL databases with sharding (e.g. MongoDB relies on a star of cliques).

3. OUR APPROACH

Our framework comprises: (i) a component library, (ii) a DSL, and (iii) a runtime. The DSL is simple and expressive enough to describe a large array of topologies that can be difficult to achieve with earlier methods. It achieves this goal by allowing developers to construct a complex topology by assembling simpler blocks, termed *components*. To support this process, our framework provides a component library that includes, by default, base components that implement basic topology shapes such as rings, grids, etc. Finally, our framework comes with a runtime that handles under-the-hood the role allocation and the differentiation of nodes that belong to different components.

3.1 Component library

In our proposal, a component is a subset of message-passing *nodes* organized in a particular *elementary topology*. The component library contains a predefined set of components implementing a range of such elementary topologies (a ring, a tree, a torus), that a developer can combine to build a complex distributed topology. This combination relies on *ports* and *links*. Ports are logical point of contact for a given component and links are logical connections between two components (through ports). At runtime, a port is managed by (at least) one node in the corresponding component, and at the node level, a link is a connection between two nodes from two different components.

From an implementation point of view, components and links are implemented using multiple *layers* of overlays that are built upon each other: one self organizing overlay per component (known as the component's *core protocol*) realizes the component's actual shape, while two other overlays are used to locate ports, and realize links. The system's *resulting overall topology* is the union of these different overlays.

3.2 DSL

In order to globally describe a *target topology* without bothering with the low-level, local behavior of individual nodes, the framework provides a very basic DSL used to write the configuration file that will be interpreted by the runtime. The key elements of this DSL/configuration file are: (i) a list of the basic shapes (each represented by a component) involved in the overall topology, and some rules to decide which node will be assigned to which component; (ii) for each component, a list of the ports it provides, and some rules to decide which node(s) will take in charge each port; (iii) a set of links between ports, represented as a list of pairs of ports.

The superposition of these three elements (components, ports for each component, links between ports) completely defines a target topology and enables the description of a large array of complex topologies similar to those used in today's real world applications.

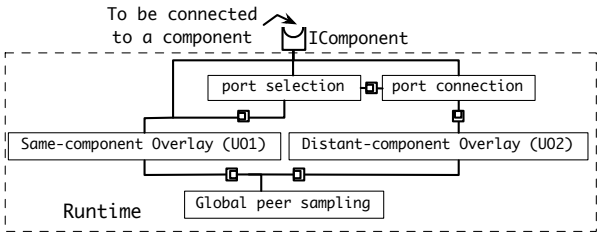

Figure 1: Organization of the runtime

3.3 Runtime implementation

The translation from the high-level target topology description to the actual low-level behavior of each individual node – and all the tedious details such as node assignment, port management, link establishment, ... – are to be handled by a runtime. We propose an implementation of this runtime as a set of gossip greedy optimization sub-procedures realizing different layers of overlays, described in Figure 1. Gossip algorithms are probabilistic, naturally resilient and offer good convergence times in most practical situations, with theoretical convergence guarantees under stable conditions. Two *utility* overlays (UO1 and UO2) are in charge of assigning nodes to each component, gather nodes from the same component and maintaining "long distance" connections between nodes from different components (for performance issues). Two additional overlays handle the mapping between logical ports and actual nodes (port selection) and the connection between different ports according to the links specified in the target topology.

4. EVALUATION

The goal of our evaluation is to show the applicability of our approach. We realized a proof-of-concept implementation of the runtime described in section 3.3. We also used the overlay-building algorithm Vicinity [23] to create a few basic shape components (Ring, Star, Clique) for the library described in 3.1. We then used them to show that our approach: (a) can actually generate complex topologies, comparable to those used currently in real-world applications; (b) is easy to use; (c) is efficient, i.e. doesn't generate an unreasonable overhead and converges fast enough.

All experiments were run in the PeerSim simulator [17] and we used the simulator configuration file as a substitute to the DSL we described above. All measures were averaged over 25 runs, to smooth the noise due to the probabilistic nature of gossip algorithms. We computed 90% confidence intervals but they were negligible and we do not display them.

We ran various experiments: (i) building various topologies comparable to those used in real world applications; (ii) convergence speed for the different sub-procedures of our framework in a Ring of Rings topology; (iii) ability to dynamically reconfigure in presence of evolving needs; (iv) scalability in terms of total number or nodes (logarithmic, Fig. 2) and in terms of number of components (linear, Fig. 3); (v) bandwidth consumption of the framework runtime, relative to the bandwidth needed to realize basic shapes (Fig. 4).

Detailed results are omitted here for lack of space, but results are encouraging and conform with the existing literature about gossip protocols and self-organizing overlays.

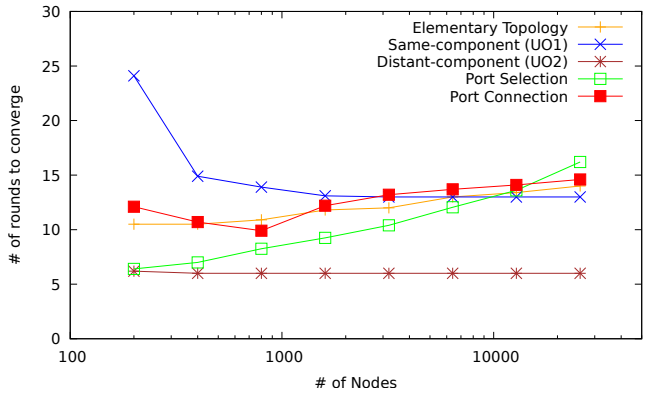

Figure 2: Convergence time of the various sub-procedures for a system of 20 components. It is fast and scales well with the number of nodes.

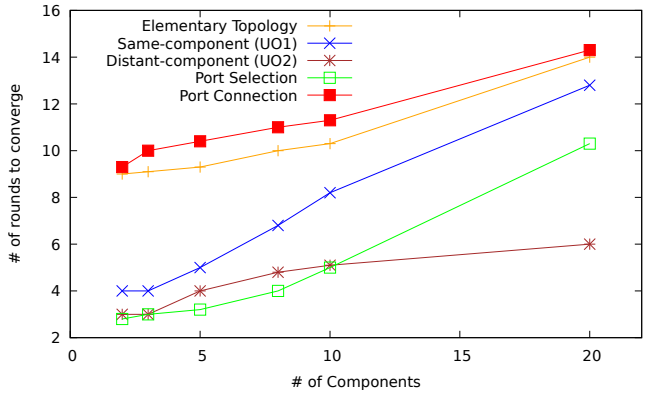

Figure 3: Convergence time of the various sub-procedures for a system of 25600 nodes. It is fast and increases slowly with the number of components.

5. CONCLUSION & FUTURE WORK

We proposed a programming framework constituted of a DSL, a component library and a runtime that enables developers to define and maintain complex target topologies by assembly of simpler shapes. We further demonstrated that our approach can be efficient and scalable.

We could push our work further in (at least) two different directions: (i) add more features to our proposed framework, develop a more complete and efficient tool-chain, and transform it into a real, production-ready product; or (ii) apply self-organization and composition to other problems in distributed systems.

In particular, there are many opportunities to leverage *opportunistic* composition across initially unrelated services to provide better Quality of Service (QoS) and ensure some non functional properties (better latency, load repartition, etc.), especially in the emerging Internet of Things (IoT). We can even imagine that a group of nodes could leverage a third-party system as relays and use it to remain connected.

However, this requires a common framework and new tools to be put in place to *detect* and *evaluate* such composition opportunities, and to enable communication and coopera-

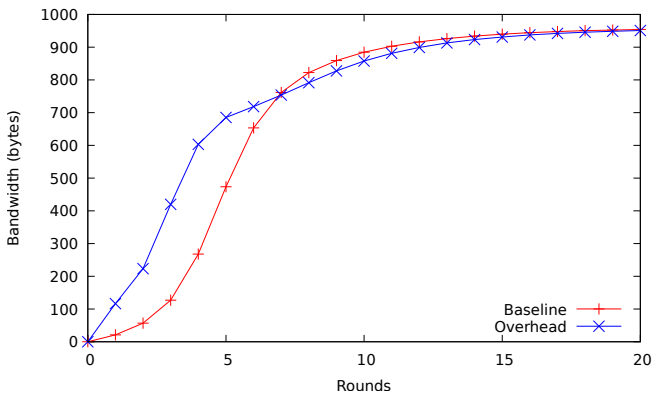

Figure 4: Comparison of bandwidth consumption (in bytes) between the core protocol and our runtime's sub-procedures, for a system of 20 components and 25,600 nodes. Both follow the same pattern, and both are very small.

tion between unrelated systems who have no prior knowledge of each other.

6. ACKNOWLEDGMENT

This thesis is funded by the French Ministry of Higher Education and Research and is supervised by François Taiani and Yérom-David Bromberg from the ASAP project-team at IRISA/Inria Rennes – Bretagne Atlantique.

7. REFERENCES

[1] R. Baraglia, P. Dazzi, M. Mordacchini, and L. Ricci. A peer-to-peer recommender system for self-emerging user communities based on gossip overlays. *Journal of Computer and System Sciences*, 79(2):291–308, 2013.

[2] M. Bertier, D. Frey, R. Guerraoui, A.-M. Kermarrec, and V. Leroy. The gossple anonymous social network. In *Middleware*, 2010.

[3] G. Blair, Y.-D. Bromberg, G. Coulson, Y. Elkhatib, L. Réveillère, H. B. Ribeiro, E. Rivière, and F. Taïani. Holons: Towards a systematic approach to composing systems of systems. In *Int. Workshop on Adaptive and Reflective Middleware*, ARM, 2015.

[4] S. Bouget, H. Kervadec, A.-M. Kermarrec, and F. Taïani. Polystyrene: The decentralized data shape that never dies. In *Distributed Computing Systems (ICDCS), 2014 IEEE 34th International Conference on*, pages 288–297. IEEE, 2014.

[5] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani. The fractal component model and its support in java. *Software: Practice ...*, pages 1257–1284, 2006.

[6] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg. Spidercast: a scalable interest-aware overlay for topic-based pub/sub communication. In *DEBS*, pages 14–25, 2007.

[7] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies*, pages 46–66, 2001.

[8] G. Coulson, G. Blair, P. Grace, F. Taiani, A. Joolia, K. Lee, J. Ueyama, and T. Sivaharan. A generic component model for building systems software. *ACM TOCS*, 26(1).

[9] L. Glendenning, I. Beschastnikh, A. Krishnamurthy, and T. Anderson. Scalable consistency in Scatter. *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 15–28, 2011.

[10] Q. Huang, K. Birman, R. van Renesse, W. Lloyd, S. Kumar, and H. C. Li. An analysis of facebook photo caching. In *SOSP*, 2013.

[11] M. Jelasity, A. Montresor, and O. Babaoglu. T-Man: Gossip-based fast overlay topology construction. *Computer Networks*, 53(13):2321–2339, Aug. 2009.

[12] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. Van Steen. Gossip-based peer sampling. *ACM TOCS*, 25(3):8, 2007.

[13] J. C. A. Leitao and L. E. T. Rodrigues. Overnesia: A resilient overlay network for virtual super-peers. In *SRDS*, 2014.

[14] G. Mega, A. Montresor, and G. P. Picco. Efficient dissemination in decentralized social networks. In *P2P*, 2011.

[15] H. Miranda, A. Pinto, and L. Rodrigues. Appia, a flexible protocol kernel supporting multiple coordinated channels. In *in Proc. 21st Int. Conf. on Dis. Comp. Sys. (ICDCS-21)*, pages 707–710. IEEE, 2001.

[16] MongoDB Inc. *MongoDB Manual (version 3.2) / Sharded Cluster Query Routing*. accessed 11 May 2016, https://docs.mongodb.com/manual/core/sharded-cluster-query-router/.

[17] A. Montresor and M. Jelasity. PeerSim: A scalable P2P simulator. In *P2P*, 2009.

[18] A. Montresor, M. Jelasity, and O. Babaoglu. Chord on demand. In *Proc. of the IEEE Int. Conf. on Peer-to-Peer Comp (P2P'05)*, pages 87–94. IEEE, August/September 2005.

[19] L. Seinturier, P. Merle, R. Rouvoy, D. Romero, V. Schiavoni, and J.-B. Stefani. A component-based middleware platform for reconfigurable service-oriented architectures. *Software: Practice and Experience*, pages n/a–n/a, 2011.

[20] B. Technologies. *Riak KV Usage Reference / V3 Multi-Datacenter Replication Reference: Architecture*. accessed 11 May 2016, http://docs.basho.com/riak/kv/2.1.4/using/reference/v3-multi-datacenter/architecture/.

[21] R. van Renesse, K. Birman, M. Hayden, A. Vaysburd, and D. Karr. Building adaptive systems using ensemble. *Softw. Prac. and Exp.*, 28(9):963–979, 1998.

[22] S. Voulgaris and M. Van Steen. Epidemic-style management of semantic overlays for content-based searching. In *Euro-Par*. 2005.

[23] S. Voulgaris and M. van Steen. Vicinity: A pinch of randomness brings out the structure. In *Middleware 2013*, pages 21–40. Springer, 2013.

[24] C. K. Yeo, B.-S. Lee, and M. H. Er. A framework for multicast video streaming over ip networks. *J. of Network and Comp. App.*, 26(3):273–289, 2003.