"An Operational Semantics for Multicasting Systems with Monotonic Values"

Haeri, Seyed Hossein ; Van Roy, Peter

Abstract

We present an operational semantics as a simplified model for edge computing: Nodes can leave and join any time; each node performs computations independently, but, can also wait for values to arrive from peers; nodes can multicast to one another; and, most importantly, values can only grow monotonically. We prove an eventual consistency result for our operational semantics.

Document type : *Communication à un colloque (Conference Paper)*

An Operational Semantics for Multicasting Systems with Monotonic Values *

Seyed H. Haeri (Hossein) and Peter Van Roy

Université catholique de Louvain, Belgium
{hossein.haeri,peter.vanroy}@uclouvain.be

Abstract

We present an operational semantics as a simplified model for edge computing: Nodes can leave and join any time; each node performs computations independently, but, can also wait for values to arrive from peers; nodes can multicast to one another; and, most importantly, values can only grow monotonically. We prove an eventual consistency result for our operational semantics.

1 Introduction

In edge computing, nodes are only loosely coupled. They are very dynamic on when they join or leave. A node, also, typically has limited computational power and resource. General computations, however, need to take place on edge devices. Yet, there are still no many formal models available which can truly describe edge computing. Reasoning about edge computing is, hence, not quite possible these days.

To that end, we present an operational semantics in this paper that is parametrised by the semantics of computations done at individual nodes. Simplicity is a key feature of our operational semantics that facilitates reasoning. Nodes, in our model, can leave and join at any time. Each node can contribute their own part to a general computation or wait for peers to come back with the results of their own part of computation. At each step, each node receives a new instruction. When to take each instruction is a property of each node that is only deterministic to the node itself. Besides, each node can choose to multicast values to peers. Values are only allowed to increase monotonically so that need rises for synchronisation.

Under certain common fairness properties, we show how every set of nodes is eventually consistent: Theorem 1. Due to space restrictions, we drop technicality – especially, on the proof of Theorem 1.

The research in this paper is amongst the last steps taken over the SyncFree project. The need for this research was clearly indicated over the last year of SyncFree. The research in the paper is to be continued over a follow-up H2020 project called LightKone that has only very recently been accepted. This paper and its accompanying talk are to welcome discussion and feedback on our early developments.

2 Settings

ν ranges over nodes. We take the syntax of expressions (ranged over by $e, e', \ldots, e_1, e_2, \ldots$) and values (ranged over by $v, v', \ldots, v_1, v_2, \ldots$) for granted. We single out a class of values $\circ$ for expressions with *lack*ing variables. (See below for more.) The subset $\circ/x$ of $\circ$ represents those elements of the latter set specifically lacking x, over which $\circ_x$ ranges.[1] Instructions are

*Funded by the Free project of the European Seventh Framework Programme under Grant Agreement 609551.

[1] Although similar to Nominal Set Theory [6], this is different because $\circ_x$ has a structure too.

ranged over by ι taking the forms $\mathsf{run}\ e \mid \mathsf{bind}\ x\ \mathsf{to}\ v \mid \mathsf{mcast}\ x\ \mathsf{to}\ \{\overline{\nu}\}$. Let I range over sets of instructions. The instruction that corresponds to ν in I is denoted by I_ν. N ranges over sets of nodes (denoted by $\{\overline{\nu}\}$) that we call hordes.

Each node ν has a *configuration* $c(\nu)$ that consists of the quadruple $\langle T, W, B, S\rangle$, for *ToTake*, *WaitFor*, *Broadcast* repository, and *Store*, respectively: the instructions for a node to do; the expressions waiting for the arrival of new information; the bindings broadcast (by peers in the same horde); and, a mapping of variables to values. We write $c_N(\nu)$ for ν's configuration when $\nu \in N$. When a node ν evolves from a configuration c_1 to a configuration c_2, we write $\nu \vdash c_1 \to c_2$. When the node of discourse is clear, we drop the "$\nu \vdash$" portion and just write $c_1 \to c_2$. We break the monolithic description of a node evolution $\langle T, W, B, S\rangle \to \langle T', W', B', S'\rangle$ into the collection of four smaller elements $\{T \to T', W \to W', B \to B', S \to S'\}$, each element of which we will display in a separate line. Inspired by I-MSOS [5], that gives us the freedom of leaving out the elements of a configuration that remain untouched over a node evolution.

We assume the availability of an expression evaluation "$\Downarrow$" with the schema $S_e : e \Downarrow S_v : v$, reading: 'In a store S_e, the expression e reduces to the value v with the store S_v that is possibly updated over the evaluation.' The value produced by $\Downarrow$ can also be $\circ_x$, in which case, $\Downarrow$ signals it that a new binding for x is required for it to continue the evaluation. With the complexity allowed for $\circ_x$, we reserve $r(\circ_x)$ for $\circ_x$ modified so it is known to $\Downarrow$ how to *resume* the original computation from where left. A necessary condition for $\Downarrow$ is that it can only modify bindings **monotonically** over expression evaluation. Note, then, that it is permissible for $S_e : e \Downarrow S_v : v$ that $dom(S_v) \supset dom(S_e)$. Another necessary condition is that the exact same "$\Downarrow$" should be provided to each and every node. Finally, it should be that "$\Downarrow$" is deterministic in the following sense [3, 4]: $\forall S_e, e.\ [(S_e : e \Downarrow S_v : v) \wedge (S_e : e \Downarrow S'_v : v')] \Rightarrow [S_v = S'_v \wedge v = v']$.

We use the notation $T + \iota$ for T augmented with the instruction ι. When over an evolution $T + \iota \to T$, it is the task ι that has had the top priority and that is done over the evolution. Note that multiple instances of a single instruction might coexist in a single T. When $c(\nu) = \langle T, W, B, S\rangle$, we write $c(\nu) + \iota$ for $\langle T + \iota, W, B, S\rangle$

3 Nodes and Hordes

The node operational semantics is defined by the rules of Fig. 1, where the schema is $\nu \vdash c_1 \to c_2$. We call the the transition from c_1 to c_2 an action [1]. Let $\alpha, \alpha', \ldots, \alpha_1, \alpha_2, \ldots$ range over actions. When ν takes the action α to get from c_1 to c_2, we write $\nu \vdash c_1 \xrightarrow{\alpha} c_2$. When an instance of a rule (R) is used for the action α to take place, we write $\alpha \in$ (R). When $\alpha \in$ (E-Done) or $\alpha \in$ (E-Susp), we write $\alpha \in$ (E-*). We use a similar convention for (B-*) and (N-*).

Similar to Bloom^L [2], we model the evolution of a system using the concept of timesteps: Let N and N' be hordes. Write $N \overset{I}{\Rightarrow} N'$ – read N evolves to N' – when: (i) $|N| = |I|$, and (ii) $\forall \nu \in N \cap N' \exists \alpha_\nu.\ \nu \vdash c_N(\nu) + I_\nu \xrightarrow{\alpha_\nu} c_{N'}(\nu)$. We drop I when not required and write $N \Rightarrow N'$. For $\#\overline{I} = n > 0$, we write $N \overset{\overline{I}}{\Rightarrow}{}^* N'$ for $N = N_0 \overset{I_1}{\Rightarrow} \ldots \overset{I_n}{\Rightarrow} N_n = N'$. When $\overline{I}$ is not important in a context, we drop it and write $N \Rightarrow^* N'$. In such a case, we write $\#(N \Rightarrow^* N')$ for $\#\overline{I}$.

Let $\nu \in N$ and $\nu' \in N'$. Suppose also that $c_N(\nu) = \langle \_, \_, \_, S\rangle$ and $c_{N'}(\nu') = \langle \_, \_, \_, S'\rangle$. Define $\Delta_{N,N'}(\nu, \nu') = \{x \in dom(\nu) \cap dom(\nu') \mid S(x) \neq S'(x)\}$. Let $\Delta(N, N') = \sum_{\nu \in N, \nu' \in N'} \Delta_{N,N'}(\nu, \nu')$. When $N = N'$, we simply write Δ_N. Moreover, for the suitable k, we write Δ_k for Δ_{N_k}.

We call an action α generative when: $\alpha \in$ (N-*); or, $\alpha \in$ (E-*) and α modifies store. Call α nongenerative otherwise. When every action in a horde evolution $N \Rightarrow N'$ is nongenerative and $N' \subseteq N$, we call the evolution itself nongenerative. Likewise, call $N \Rightarrow^* N'$ nongenerative

$$\frac{S_e : e \Downarrow S_v : v}{\begin{cases} T + \mathsf{run}\ e \to T \\ S_e \to S_v \end{cases}}\ \text{(E-Done)} \qquad \frac{S_e : e \Downarrow S_\circ :^{\circ_x}}{\begin{cases} T + \mathsf{run}\ e \to T \\ W \to W +^{\circ_x} \\ S_e \to S_\circ \end{cases}}\ \text{(E-Susp)}$$

$$\frac{x \notin dom(S) \quad {\circ_x} \notin W}{\begin{cases} (B, x \mapsto v) \to B \\ S \to (S, x \mapsto v) \end{cases}}\ \text{(B-NW)} \qquad \frac{S(x) = v \quad W = W' +^{\circ_x}}{\begin{cases} (B, x \mapsto v') \to B \\ S \to S[x \mapsto v \sqcup v'] \\ W \to W' \\ T \to T + \mathsf{run}\ r({\circ_x}) \end{cases}}\ \text{(B-Wait)}$$

$$\frac{x \notin dom(S) \quad {\circ_x} \notin W}{\begin{cases} T + \mathsf{bind}\ x\ \mathsf{to}\ v \to T \\ S \to (S, x \mapsto v) \end{cases}}\ \text{(N-NW)} \qquad \frac{x \notin dom(S) \quad W = W' +^{\circ_x}}{\begin{cases} T + \mathsf{bind}\ x\ \mathsf{to}\ v \to T + \mathsf{run}\ r({\circ_x}) \\ S \to (S, x \mapsto v) \\ W \to W' \end{cases}}\ \text{(N-Wait)}$$

$$\frac{\begin{array}{c} \nu \in N \quad \{\overline{\nu}\} \subseteq N \quad \nu' \in \{\overline{\nu}\} \\ v = S_\nu(x) \qquad v' = B_{\nu'}(x) \end{array}}{\begin{cases} \nu \vdash T_\nu + \mathsf{mcast}\ x\ \mathsf{to}\ \{\overline{\nu}\} \to T_\nu \\ \nu' \vdash B_{\nu'} \to B_{\nu'}[x \mapsto v \sqcup v'] \end{cases}}\ \text{(MCast)}$$

Figure 1: The Node Operational Semantics

when, for every $1 \leq i \leq n$, the evolution $N_{i-1} \Rightarrow N_i$ is nongenerative, where $n = \#(N \Rightarrow^* N')$.

For a node ν, write $m_\nu(x) = m$ when x is guaranteed to be multicast at least once every m actions at ν. When no such guarantee is at hand for ν, we write $m_\nu(x) = \aleph_0$. Write $r_\nu(x) = r$ when x is guaranteed to be committed at least once every r actions at ν. We use $r_\nu(x) = \aleph_0$ similar to $m_\nu(x)$. Call a node ν fair on x when $m_\nu(x) < \aleph_0$ and $r_\nu(x) < \aleph_0$. Write $p_\nu(\nu') = p$ when ν is guaranteed to multicast to a peer ν' at least once every p actions. Call a horde N all-fair if every node $\nu \in N$ is fair on every variable and $p_\nu(\nu') < \aleph_0$ for every $\nu' \in N \setminus \{\nu\}$.

Theorem 1 (Eventual Consistency)**.** *Let N be an all-fair horde. Suppose that $N \Rightarrow^* N'$ be nongenerative and $n = \#(N \Rightarrow^* N')$. Then,* $\lim_{n\to\infty} \Delta_n = \varnothing$.

References

[1] A. Bouajjani, C. Enea, and J. Hamza. Verifying Eventual Consistency of Optimistic Replication Systems. In S. Jagannathan and P. Sewell, editors, *POPL*, pages 285–296. ACM, Jan 2014.

[2] N. Conway, W. R. Marczak, P. Alvaro, J. M. Hellerstein, and D. Maier. Logic and Lattices for Distributed Programming. In M. J. Carey and S. Hand, editors, *SCC*, page 1, Oct 2012.

[3] S. H. Haeri. Observational Equivalence and a New Operational Semantics for Lazy Evaluation with Selective Strictness. In Z. Majkic, S.-Y. Hsieh, J. Ma, I. M. M. El Emary, and K. S. Husain, editors, *TMFCS*, pages 143–150. ISRST, Jul 2010.

[4] S. H. Haeri and S. Schupp. Distributed Lazy Evaluation: A Big-Step Mechanised Semantics. In 1^{st} *4PAD*, pages 751–755. IEEE, Feb 2014.

[5] P. D. Mosses and M. J. New. Implicit Propagation in Structural Operational Semantics. *ENTCS*, 229(4):49–66, 2009.

[6] A. M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science.* Cambridge University Press, New York, NY, USA, 2013.