Temporal Reasoning about Resources for Deadline Assurance in Distributed Systems

Xinghui Zhao
Department of Computer Science
University of Saskatchewan
176 Thorvaldson Building, 110 Science Place
Saskatoon, Canada, S7N 5C9
Email: xinghuizhao@agents.usask.ca

Nadeem Jamali
Department of Computer Science
University of Saskatchewan
176 Thorvaldson Building, 110 Science Place
Saskatoon, Canada, S7N 5C9
Email: jamali@agents.usask.ca

Abstract—In an open distributed system, computations can be carried out without statically owned resources, harnessing the collective compute power of the resources connected by the Internet. However, realizing this potential requires efficient and scalable resource discovery, coordination and control, which present challenges in a dynamic, open environment. We present ROTA, a resource-oriented temporal logic, which addresses these challenges by enabling computations to reason about future availability of resources. In ROTA, computational resources are defined over time and space, and represented using resource terms, which specify key attributes of resources. Syntax and semantics of ROTA are described. Theorems are derived to illustrate how the logic can be used to express resource properties of the system in time and space, track resource utilization, and calculate future availability of resources. Particularly, at any time, given a computation, it is possible to evaluate whether its deadline constraint can be assured by the available resources.

I. INTRODUCTION

With the growing ubiquity of networked computers, there is an ever increasing potential for executing computations by utilizing distributed peer-owned resources. As a result, new computation paradigms – such as grid and cloud computing – have emerged, where distributed resources are provided to various applications over the Internet. However, accurately reasoning about resource availability on a network of peer-owned resources – necessary for assurance of resource delivery – remains a significant challenge. The challenge is even more pronounced when these computations are executing in open distributed environments, because in such a context, resources can dynamically join or leave the system at any time (or can be discovered at runtime). The dynamicity that makes opportunities visible at runtime also leads to uncertainty about continuous availability of the needed resources. Meeting these challenges can be helped by computations’ ability to reason about future availability of resources.

Reasoning about resource-bounded computations has received a significant amount of attention recently [3], [10], [9], [2]: however, the emphasis has been on adapting behaviors of computations as they try to adapt to resource bounds, rather than empowering computations with the reasoning ability to better navigate in the space of resource uncertainty in search of new resources – to seek out new frontiers, in a manner of speaking. We are interested in answering the question: “Can we know at time T whether a distributed multi-agent computation A can complete its execution by deadline D?” ROTA addresses this challenge in the context of open distributed systems by providing a reasoning scheme for accurately predicting resource availability in the future.

ROTA is a resource-oriented logic for reasoning about feasibility of carrying out deadline-constrained distributed computations in resource bounded open distributed systems. ROTA focuses on computational resources – defined in a broader sense to also include communication resources required by distributed computations – consuming which enables computations to make progress. In ROTA, computational resources are defined over time and space. Resources are represented using resource terms, which contain several key attributes of the specified resources: type, density, time and location of existence.

Because resources are the focus of this work, we represent computations which seek to use resources in terms of the resources they require. In other words, we will be interested in which resources, when and how much of them do computations consume, rather than what the computations do while consuming them. Specifically, each computation is represented by its resource requirements. ROTA uses labeled transition rules to describe the process of resource consumption, using which we illustrate how resource availability over the course of a computation can be reasoned about. Representing computations in this manner together with the resource availability over time makes it possible to accurately reason about whether the deadline constraint of a distributed computation can be assured.

The primary contribution of this paper is in reifying resources and resource requirements of distributed computations in a novel way, and then developing a logic for reasoning about accommodation of new computations.

The rest of the paper is organized as follows. We review related work in Section II; Section III shows how resources are represented in ROTA; Section IV gives the representation of distributed computations in ROTA; formal definition of ROTA, including syntax and semantics, is presented in Section V; and Section VI concludes the paper and gives future directions of
the work.

II. RELATED WORK

Limited resource has always been a major concern in concurrent systems [6]. As for resource reasoning, significant work has been done in the area of multi-agent planning and formal logic. Horvitz’s work [11] is the first to address the concern about limitations of computational resource in the environment where reasoning systems execute. Since then, the boundedness of computational resources has received increasing attention in reasoning about multi-agent behaviours [9].

ParcPlan [7] integrates temporal reasoning with resource reasoning, and resource feasibility is determined by checking the resource capacity constraint at starting points of resource requests. TRP [5] formalizes resource constraints in CSP (constraint satisfaction problem) terms, and propagation techniques are provided to synthesize new temporal constraints by reasoning on resource representation. Unlike parcPlan and TRP, where temporal and resource reasoning is performed after a plan has been obtained, realPlan [13] separates resource reasoning (scheduling) from causal reasoning (planning), leading to improved planning performance.

To the best of our knowledge, in most of the existing work in resource-bounded agents reasoning, the emphasis has been on the behaviors of agents/computations constrained by resource bounds, rather than empowering computations with the reasoning ability to better navigate in a world of resource uncertainty. ROTA enables computations to reason about resource availability in the future, so that the feasibility of a computation plan (whether or not a specific computation can be completed by its deadline) can be determined well in advance.

Besides the multi-agent planning approaches, in the area of formal logic, step logic [8] was first to represent resource/time in reasoning systems. However, step logic does not have adequate semantics. Timed Reasoning Logics (TRL) [3], which is inspired by step logic, provides a complete and decidable context-logic style formalism to reason about time-bounded reasoners. The limitation of these approaches is that resources are represented by some count, and usually only one type of resource is considered. BMCL [2] is the first attempt to integrate multiple resources in one reasoning system. However, resources are not explicitly represented in BMCL. Instead, resource bounds are expressed as axioms in the logic, so that during the course of reasoning, the resource bounds can be verified. Therefore, BMCL is not capable of reasoning about the resource situation in the system. Unlike the above approaches, ROTA is a resource-oriented logic, and it explicitly represents multiple types of resource in a uniform way using resource terms. In ROTA, the availability of resources throughout the course of computation can be easily reasoned about.

III. RESOURCE REPRESENTATION

Distributed computations execute in environments where computational and communication resources are spread over time and space. In an approach inspired by the CyberOrgs model for resource bounded concurrent systems [12], we define resources in time and space. Specifically, each computational resource is represented by a resource term: \([\tau]_{\xi}\), where \(\tau\) represents the rate of availability of the resource, \(\xi\) denotes the located type of the specified resource, which contains both the type of the resource and the location where the resource is residing. For example, for “CPU resource on location \(l_1\)” the located type is \(\langle\text{cpu}, l_1\rangle\). In comparison, the spatial information for a network resource has to identify both the source and destination nodes of the resource. For example, located type of a network resource that can be used to send data from location \(l_1\) to \(l_2\) would be specified as \(\langle\text{network}, l_1 \rightarrow l_2\rangle\).

Because each resource term is associated with a time interval \(\tau\), relationships between time intervals must be defined before we can discuss the operations on resource terms. In ROTA, we use Interval Algebra [4] to formalize relations between two time intervals. As shown in Table I, the seven possible relations (or thirteen if we count the inverse relations) are before (\(<\)), equal (\(=\)), during (\(\in\)), meets (\(\cap\)), \(\^\leftarrow\) overlaps (\(\cup\)), starts (\(\sqsubset\)), and finishes (\(\sqsubset\)).

<table>
<thead>
<tr>
<th>Relation</th>
<th>Interpretation</th>
<th>Illustration</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\tau_1 &lt; \tau_2)</td>
<td>(\tau_1 \text{ before } \tau_2)</td>
<td>(\tau_1 \tau_2 \tau_1 \tau_2)</td>
</tr>
<tr>
<td>(\tau_1 &gt; \tau_2)</td>
<td>(\tau_1 \text{ after } \tau_2)</td>
<td>(\tau_1 \tau_2 \tau_1 \tau_2)</td>
</tr>
<tr>
<td>(\tau_1 = \tau_2)</td>
<td>(\tau_1 \text{ equals } \tau_2)</td>
<td>(\tau_1 \tau_2 \tau_1 \tau_2)</td>
</tr>
<tr>
<td>(\tau_1 \in \tau_2)</td>
<td>(\tau_1 \text{ during } \tau_2)</td>
<td>(\tau_1 \tau_2 \tau_1 \tau_2)</td>
</tr>
<tr>
<td>(\tau_1 \cap \tau_2)</td>
<td>(\tau_1 \text{ meets } \tau_2)</td>
<td>(\tau_1 \tau_2 \tau_1 \tau_2)</td>
</tr>
<tr>
<td>(\tau_1 \cap \tau_2)</td>
<td>(\tau_1 \text{ overlaps } \tau_2)</td>
<td>(\tau_1 \tau_2 \tau_1 \tau_2)</td>
</tr>
<tr>
<td>(\tau_1 \sqsubset \tau_2)</td>
<td>(\tau_1 \text{ starts } \tau_2)</td>
<td>(\tau_1 \tau_2 \tau_1 \tau_2)</td>
</tr>
<tr>
<td>(\tau_1 \sqsubset \tau_2)</td>
<td>(\tau_1 \text{ finishes } \tau_2)</td>
<td>(\tau_1 \tau_2 \tau_1 \tau_2)</td>
</tr>
</tbody>
</table>

Each time interval \(\tau\) has a start time \(t_{\text{start}}\), and an end time \(t_{\text{end}}\). In this paper, we also use \((t_{\text{start}}, t_{\text{end}})\) as an alternative notation for time interval \(\tau\). Furthermore, binary operations on sets, such as union (\(\cup\)), intersection (\(\cap\)), relative complement (\(\setminus\)) are also available for time intervals.

Resources in a distributed system can be represented by a set of resource terms, each with its own located type. Resources joining or leaving the system can then be expressed by union and relative complement operations on resource sets, respectively.

If two resource terms in a resource set have the same located types and overlapping time intervals, they can be combined by

1. The product \(\tau \times \xi\) gives the total quantity of the available resource over the course of time interval \(\tau\).
2. \(\tau_1 \cap \tau_2\) means \(\tau_2\) starts immediately after \(\tau_1\) ends.
3. \(\tau_1 \cup \tau_2\) means \(\tau_1\) and \(\tau_2\) start at the same time point.
4. \(\tau_1 \setminus \tau_2\) means \(\tau_1\) and \(\tau_2\) end at the same time point.
a process of simplification, where for any interval for which they overlap, their rates are added, and for remaining intervals, they are represented separately in the set:

$$\{r_1\}_{\xi}^{T_1} \cup \{r_2\}_{\xi}^{T_2} = \{r_1\}_{\xi}^{T_1 \cap T_2}, \{r_2\}_{\xi}^{T_1 \cap T_2}, \{r_1 + r_2\}_{\xi}^{T_2}\}

The simplification essentially aggregates identical located type resources available simultaneously, which can lead to a larger number of terms. Resource terms can reduce in number if two identical located type resources with identical rates have time intervals that meet.

Note that if the time interval of a resource term is empty, the value of the resource term is 0, or null. In other words, resources are only defined during non-empty time intervals.

The notion of negative resource terms is not meaningful in this context; so, resource terms cannot be negative. We define an inequality operator to compare two resource terms, from the perspective of a computation’s potential use of them. We say that a resource term is greater than another if a computation that requires the latter, can instead use the former, with some to spare. We specifically state it as follows: $\{r_1\}_{\xi}^{T_1} > \{r_2\}_{\xi}^{T_2}$ if and only if $\xi_1 \geq \xi_2$, $r_1 > r_2$, and $T_2 \cap T_1$. Note that it is not necessarily enough for the total amount of resource available over the course of an interval to be greater. Consider a computation that is able to utilize needed resources only during interval $T_2$; if additional resources are available outside of $T_2$, but not enough during $T_2$, it does not help satisfy the computation.

The relative complement of two resource sets $\Theta_1 \setminus \Theta_2$ is defined only when for each resource term $\{r_2\}_{\xi}^{T_2}$ in $\Theta_2$, there exists a resource term $\{r_1\}_{\xi}^{T_1} \in \Theta_1$, such that $\{r_1\}_{\xi}^{T_1} > \{r_2\}_{\xi}^{T_2}$. The relative complement of two resource sets is defined as follows:

$$\Theta_1 \setminus \Theta_2 = \{r_1\}_{\xi}^{T_1} - \{r_2\}_{\xi}^{T_2} \cup \Theta_1 \setminus \Theta_2$$

where $\{r_1\}_{\xi}^{T_1} - \{r_2\}_{\xi}^{T_2} = \{r_1\}_{\xi}^{T_1 \setminus T_2}, \{r_1 - r_2\}_{\xi}^{T_2}\}$.

Following are some examples of calculations on resource sets.

$$\{5\}_{cpu,l_1}^{(0,3)} \cup \{5\}_{network,l_1 \rightarrow l_2}^{(0,5)} = \{5\}_{cpu,l_1}^{(0,3)} \cup \{5\}_{cpu,l_1}^{(0,5)} \cup \{5\}_{network,l_1 \rightarrow l_2}^{(0,5)}$$

$$\{5\}_{cpu,l_1}^{(0,3)} \cup \{5\}_{cpu,l_1}^{(0,5)} = \{10\}_{cpu,l_1}^{(0,3)} = \{5\}_{cpu,l_1}^{(3,5)}$$

$$\{5\}_{cpu,l_1}^{(0,3)} \cup \{3\}_{cpu,l_1}^{(1,2)} = \{5\}_{cpu,l_1}^{(0,1)} \cup \{2\}_{cpu,l_1}^{(1,2)} \cup \{5\}_{cpu,l_1}^{(2,3)}$$

Union and relative complement operations on resource sets allow modeling of resources that join or leave the system dynamically, as typically happens in open distributed systems such as the Internet.

IV. COMPUTATION REPRESENTATION

A computation consumes resources at every step of its execution. We abstract away what a distributed computation does and represent it by the resource requirements for each step of its execution; this idea is inspired by the CyberOrgs [12] model.

A. Actor Computations

We think of distributed computations as computations carried out by actors [1], which are autonomous concurrently executing active objects which communicate with each other using buffered, asynchronous, point-to-point messages. Actors have globally unique names, and maintain queues of unprocessed messages they have received, which are processed in the order of their arrival. Actors carry out the computations specified by their behaviors (i.e., methods) in the course of processing messages. An actor may evaluate expressions, send messages to other actors, create a finite number of new actors with some predefined behaviours, or change its own state and become ready to process the next message. In addition, in a distributed execution environment, an actor may use a fourth primitive migrate in order to migrate to another location, and continue execution there. In other words, an actor’s behaviour is a sequence of these five types of actions. An actor utilizes processor and network resources in order to carry out these actions.

Consider a function $\Phi$, which when provided as parameters an actor’s uniquely identifying name, and the computation it is to perform, returns a set of resource amounts representing the required resources for completing the computation.\(^3\) For example, resources required for actor $a_1$—located at $l(a_1)$—to send a message $m$ to actor $a_2$—located at $l(a_2)$—is $\Phi(a_1, send(a_2, m))$. The value of a required resource is represented by $\{q\}_{\xi}$, where $q$ is the quantity of resource required, and $\xi$ is the located type. For our example, $\xi$ would be $\langle network, l(a_1) \rightarrow l(a_2) \rangle$. Natural numbers can be used for representing the quantity $q$. If actor $a_1$ needs 4 units of network resource in order to send message $m$ to actor $a_2$, then we say:

$$\Phi(a_1, send(a_2, m)) = \{4\}_{\langle network, l(a_1) \rightarrow l(a_2) \rangle}$$

Similarly, other actions of actor $a_1$, can be converted to resource amounts as well, using function $\Phi$ as follows:

$$\Phi(a_1, evaluate(e)) = \{8\}_{\langle cpu, l(a_1) \rangle}$$

$$\Phi(a_1, create(b)) = \{5\}_{\langle cpu, l(a_1) \rangle}$$

$$\Phi(a_1, ready(b)) = \{1\}_{\langle cpu, l(a_1) \rangle}$$

\(^3\)This device, although useful for simplifying our discussion, does not imply need for existence of such a function. Any-time algorithms, approximate algorithms, etc. are examples of when it is meaningful to talk about such a function. In general, at the cost of some inefficiency, estimates could be used and revised as necessary.

\(^4\)l is the location function; $l(a)$ gives the location of actor $a$
\[ \Phi(a_1, \text{migrate}(l_2)) = \{[3]_{\text{cpu}, l(a_1)}, [0]_{\text{network}, l(a_1) \rightarrow l_2}, [3]_{\text{cpu}, l_2} \} \]

The quantities 4, 8, 5, 1, etc., in the above equations are hypothetical amounts used for illustration purposes. Note that a single actor action may require multiple types of resources. For instance, the migrate operation needs both CPU and network resources, because in order to be able to resume remotely in its current state, the migrating actor needs first to be serialized, then sent to the destination node over the network, and finally unserialized at the destination node to resume execution.

We use a sequence of these resource requirements to refer an actor. Specifically, the computation to be carried out by an actor \( a \) is represented by a sequence of resource amounts, each amount identifying resources required by a particular action.

Furthermore, an individual actor’s computation is sequential, which means that actions must be taken in a specific order. Consequently, an action may not be available for execution unless all previous actions have been completed.

Here, we define the notion of a possible action.

**Definition 1: Possible Action** An actor action \( \gamma \) is a possible action if and only if one of the following is true:

- \( \gamma \) is the first action of the actor,
- all actions which precede \( \gamma \) in the sequence of the actor’s actions have already been completed.

The necessary and sufficient condition for an actor action to be completed can now be stated as follows:

**Axiom 1** An action \( \gamma \) of actor \( a \), can be completed if and only if it is a possible action of \( a \), and its required resources \( \Phi(a, \gamma) \) are available.

This axiom serves as the foundation for the reasoning made possible by ROTA.

**B. Resource Requirements of Distributed Computations**

We represent a distributed computation by a triple \((\Lambda, s, d)\), where \( \Lambda \) is a representation of the computation, \( s \) is the earliest start time of the computation, and \( d \) is the deadline by which the computation must complete. Particularly, the computation does not seek to begin before \( s \) and seeks to be completed before \( d \). For this to happen, \( \Lambda \) requires the resources for completing this computation – however distributed over the course of the interval – during the interval.

If there are multiple (possibly concurrent) actor computations in \( \Lambda \), we use \( \Gamma \) to denote an actor computation, and use \( \gamma \) to denote a single action of the actor (possibly carried out concurrently with other actors carrying out other actions).

1) **Single (actor) action:** We represent the resource requirements of an actor’s action as a simple resource requirement \( \rho \) defined as follows:

\[ \rho(\gamma, s, d) = [\Phi(a, \gamma)]^{(s,d)} \]

This simple resource requirement specifies the total amount of resource required for action \( \gamma \) during the time interval \((s, d)\). For convenience, we define a function \( f \), which takes as parameters a resource set \( \Theta \) and a simple resource requirement \( \rho(\gamma, s, d) \), and returns a boolean value true or false, indicating whether or not the simple resource requirement can be satisfied given the available resource set \( \Theta \):

\[ f(\Theta, \rho(\gamma, s, d)) = \bigcup_{s}^{d} \Theta \geq \Phi(\gamma) \]

where \( \bigcup_{s}^{d} \Theta \) gives the union of all resources in \( \Theta \) which exist in the interval \((s, d)\).

The following theorem states when an actor action can be completed by its deadline.

**Theorem 1: Single Action Accommodation** A computation \((\gamma, s, d)\) which only contains a single actor action \( \gamma \) can be accommodated by a system, if, by time \( s \), \( \gamma \) is a possible action, and the system satisfies the simple resource requirement \( \rho(\gamma, s, d) \): \( f(\Theta, \rho(\gamma, s, d)) = \text{true} \), where \( \Theta \) is the available resources of the system.

**Proof.** If \( f(\Theta, \rho(\gamma, s, d)) = \text{true} \), the resources required for the computation \( \Phi(\gamma) \) are available during the time interval \((s, d)\). In that case, according to Axiom 1, the computation can be completed during \((s, d)\). This proves “if”.

If the computation \((\gamma, s, d)\) can be completed, \( \gamma \) must be a possible action; otherwise, according to Axiom 1, it could not completed. Furthermore, there must be enough resource for the execution of \( \gamma \), meaning \( f(\Theta, \rho(\gamma, s, d)) = \text{true} \). This proves “only if”. \( \square \)

2) **Sequential (actor) computation:** An actor’s resource requirements – represented by \( \rho(\Gamma, s, d) \) – are for executing a sequence of actions, which may require different types of resources. Critically, the resources needed for completing an actor’s execution are required in a specific order. It is not sufficient to simply have the correct total quantities of the resources during the entire interval; the right resources are required at the right time.

We define a complex resource requirement in terms of simple resource requirements, to represent the requirements of the actor computation \( \Gamma \), as follows:

\[ \rho(\Gamma, s, d) \coloneqq \rho(\Gamma_1, s, t_1) \cup \rho(\Gamma_2, t_1, t_2) \cup \ldots \cup \rho(\Gamma_m, t_{m-1}, d) \]

where \( s < t_1 < t_2 < \ldots < t_{m-1} < d \)

\[ \rho(\gamma, s, d) \]

As shown in the above equation, function \( \rho \) breaks down the actor’s computation \( \Gamma \) into a sequence of \( m \) subcomputations. As a result, the resource requirements of the actor \((\Gamma, s, d)\) amount to a sequence of simple resource requirements for the subcomputations. Note that a sequence of actions which require the same single type of resource need not be broken down into multiple subcomputations, because this case is similar to single actor action, where having enough amount of resource during the interval will guarantee completion of the computation.

Theorem 2 states when a sequential actor computation can be completed by its deadline.
Theorem 2: Sequential Computation Accommodation

A system with resources $\Theta$ can accommodate a sequential computation $(\Gamma, s, d)$, if and only if there exist time points $t_1, t_2, \ldots, t_{m-1}$ between $s$ and $d$, which divide the time interval $(s, d)$ into a sequence of $m$ subintervals, so that the system can satisfy the simple resource requirements for each subinterval.

Proof Assume the computation $(\Gamma, s, d)$ can be accommodated by the system. We can set up break points $b_1, b_2, \ldots, b_m$ in the computation, each of which identifies the starting point of a subcomputation. In the real-time execution of the computation, the time points when those break points are encountered will be a set of $t_1, t_2, \ldots, t_{m-1}$, each of which must satisfy the corresponding simple resource requirement for the subcomputation according to Axiom 1. This proves only if.

Assume we already have a set of time points $t_1, t_2, \ldots, t_{m-1}$, for which the sequence of simple resource requirements are satisfied, according to the definition of complex resource requirement $\rho(\Gamma, s, d)$ and Axiom 1, it is obvious that the computation can be accommodated by executing each subcomputation during its corresponding time interval, during which the required resources are available. This proves if. $\square$

3) Concurrent (multi-actor) computation: A concurrent computation consists multiple actors. Here, we limit ourselves to concurrent computations involving independent actors. In other words, all actors participating in the computation are deemed to be created en masse at the beginning of the computation and actors never have to wait for messages from other actors.

Resource requirements of a concurrent computation $(\Lambda, s, d)$, can be satisfied by satisfying resource requirements requirements (defined previously) of the individual actors, as follows:

$$\rho(\Lambda, s, d) := \rho(\Gamma_{a_1}, s, d) \cup \rho(\Gamma_{a_2}, s, d) \cup \ldots \cup \rho(\Gamma_{a_n}, s, d)$$

where $\Gamma_{a_1}, \Gamma_{a_2}, \ldots, \Gamma_{a_n}$ represent computations carried out by actors $a_1, a_2, \ldots, a_n$ respectively. To simplify the model, we assume that actors do not migrate for acquiring resources. In other words, they only migrate for functional reasons. Therefore, the located type of their required resources can be easily determined.

As shown in the above equation, multiple complex resource requirements overlap on the same time interval. In order to determine whether the computation can be accommodated using available resources, we need to find an answer to the following question: “Can the system accommodate one more actor computation $\rho(\Gamma_{a_i}, s, d)$ when it has already made commitments to accommodate computations $\rho(\Gamma_{a_1}, s, d), \rho(\Gamma_{a_2}, s, d), \ldots, \rho(\Gamma_{a_{i-1}}, s, d)$?” If we can answer this question, the problem can be solved step by step, by trying to accommodate one more computation at a time. However, without a clear way of reasoning about resource consumption in the system, it is not possible to answer the question.

Next, we introduce ROTA, which provides a framework for performing such reasoning.

V. RESOURCE-ORIENTED TEMPORAL LOGIC

With resource terms/sets, ROTA is capable of describing the evolution of a distributed system by reasoning about computational resources. Furthermore, important resource-related properties can be expressed using ROTA, and propositions about deadline assurance can be verified. A formal definition of ROTA follows.

A. System Model

The ROTA system model can be represented by a 4-tuple, $M = (A, R, C, \Phi)$. $A$ is a set of actor names; $R$ is a set of resource terms; $C$ is a set of distributed computations, represented by sequences of actions taken by actors; $\Phi$ is a function which maps computations carried out by actors to the resources they require. We define $S$, the state of the system as follows:

$$S = (\Theta, \rho, t)$$

where $\Theta$ is a set of resource terms, representing future available resources in the system, starting from time $t$; $\rho$ represents the resource requirements of the computations that are accommodated by the system at time $t$; and $t$ is the point in time when the system’s state is $S$.

Progress of the system is triggered by the injection of resources. Resources specified in resource terms expire if there is no computation which requires those resources during the time intervals. This means the resources are only defined for a certain period of time, specified by the time interval in their resource terms.

If the evolution of a ROTA system is denoted by a sequence of states $(S_1, S_2, \ldots, S_n)$, the progress of the system is regulated by a labeled transition rule:

$$S_i \xrightarrow{\xi \mapsto a} S_{i+1}$$

where $\xi$ is a resource located type, and $a$ is the name of an actor. The transition rule specifies that the utilization of resource $\xi$ for actor $a$’s next action makes the system progress from state $S_i$ to the next state $S_{i+1}$. If we replace the states in the above transition rule with the detailed $(\Theta, \rho, t)$ format, the transition rule can be written as:

$$(\{[\xi]^{(t, t')}, \Theta\}, [\rho]^{(t, t')}, \rho], t) \xrightarrow{\xi \mapsto a}$$

where $[\rho]^{(t, t')}$ is the available resource of located type $\xi$, $\xi^{(t, t')}$ is the simple resource requirement of actor $a$’s action associated with resource $\xi$, and $\Delta t$ is the smallest time slice that the system can account for. Every time a transition rule is applied to the system, the system progresses one step further by time $\Delta t$. Here, the transition rule states that during the

\footnote{In practice, $\Delta t$ can be defined according to the desired control granularity.}
time interval \((t, t + \Delta t)\), the available resource \(\xi\) is used to fuel actor \(a\)'s action. As a result, by time \(t + \Delta t\), the a's requirement for \(\xi\) will be \(t \times \Delta t\) less than it was at time \(t\).

The above transition rule is the **sequential transition rule**, because only one actor in the system obtains resource and makes progress.

The sequential transition rule represents the evolution of the system when a sequential computation is carried out. However, the behaviour of a concurrent system is more interesting. In such a system, there are multiple actors sharing resources. Each actor takes a sequence of actions, which represents a sequential computation. Multiple types of resources can be consumed at the same time. The progress of a concurrent ROTA system is given by the **concurrent transition rule**:

\[
S_t \xrightarrow{\xi_1 \rightarrow a_1, \ldots, \xi_n \rightarrow a_n} S_{t+1}
\]

which is,

\[
\begin{align*}
&\left(\bigcup_{i=1}^n [t_i, t_i'] \xi_i, \Theta, \bigcup_{i=1}^n [q_i, t_i'] \xi_i, \rho, t\right) \xrightarrow{\xi_1 \rightarrow a_1, \ldots, \xi_n \rightarrow a_n} \\
&\left(\bigcup_{i=1}^n [t_i + \Delta t, t_i'] \xi_i, \Theta, \bigcup_{i=1}^n [q_i - t_i \times \Delta t, t_i'] \xi_i, \rho, t + \Delta t\right)
\end{align*}
\]

Similar to the sequential transition rule, \([t_i, t_i'] \xi_i\) is the available resource which has located type \(\xi_i\), \([q_i, t_i'] \xi_i\) is the simple resource requirement by actor \(a_i\)'s action associated with resource \(\xi_i\), and \(\Delta t\) is the smallest time slice that the system can account for.

The concurrent transition rule specifies that the system evolves by consuming multiple types of resources during one time interval \((t, t + \Delta t)\), and those resources are used to fuel multiple actors’ computations.

Note that if certain resource becomes available, yet no computations require that type of resource, the resource expires. The **resource expiration rule** is defined as follows:

\[
\left(\bigcup_{i=1}^n [t_i, t_i'] \xi_i, \Theta, \rho, t\right) \xrightarrow{\xi_k} \left(\bigcup_{i=1}^n [t_i, t_i'] \xi_i, \Theta, \rho, t + \Delta t\right)
\]

The resource expiration rule states that with time \(\Delta t\) elapsing, resource \(\xi_k\) is expired, and no computation makes any progress.

Similarly, a concurrent version of resource expiration rule is defined as follows:

\[
\begin{align*}
&\left(\bigcup_{i=1}^n [t_i, t_i'] \xi_i, \Theta, \rho, t\right) \xrightarrow{\xi_1 \rightarrow a_1, \ldots, \xi_n \rightarrow a_n} \\
&\left(\bigcup_{i=1}^n [t_i + \Delta t, t_i'] \xi_i, \Theta, \rho, t + \Delta t\right)
\end{align*}
\]

The concurrent resource expiration rule specifies the system evolution caused by multiple resources become expired.

The transition and resource expiration rules specify extreme cases where either all of the resources available at time \(t\) are consumed by actors or all of them expire. To represent a more likely scenario, the two rules can be combined to form a general transition rule, in which some resources are consumed, while others expire, as follows:

\[
\begin{align*}
&\left(\bigcup_{i=1}^n [t_i, t_i'] \xi_i, \Theta, \bigcup_{i=1}^n [q_i, t_i'] \xi_i, \rho, t\right) \xrightarrow{\xi_1 \rightarrow a_1, \ldots, \xi_n \rightarrow a_n} \\
&\left(\bigcup_{i=1}^n [t_i + \Delta t, t_i'] \xi_i, \Theta, \bigcup_{i=1}^n [q_i - t_i \times \Delta t, t_i'] \xi_i, \rho, t + \Delta t\right)
\end{align*}
\]

Besides the above transitions rules which represent system evolution over time, ROTA also has two sets of transition rules which can be applied at a time instant, representing resource acquisition and computation accommodation/leave.

In an open system, resources may join or leave the system at any time. The resource acquisition is modeled by the following **resource acquisition rule**:

\[
(\Theta, \rho, t) \xrightarrow{\Theta_{\text{join}}} (\Theta \cup \Theta_{\text{join}}, \rho, t)
\]

where \(\Theta_{\text{join}}\) is the resource set which joins the system at time \(t\). Note that there is no such a transition rule for resources leaving, because resources join only for a time interval, at the end of which they are claimed to leave the system. If a resource is going to leave the system in the future, the time of leaving must be explicitly specified at the time of joining the system.

Similar to the resources, computations in an open system may arrive or leave at any time. The transition rule for **computation accommodation** is as follows:

\[
(\Theta, \rho, t) \xrightarrow{\lambda(A, s, d)} (\Theta, \rho \cup \rho(A, s, d), t)
\]

where \(t < d\), which means that it is not possible to accommodate a computation if its deadline has passed.

Similarly, ROTA has a **computation leave** rule to represent a computation leaving the system:

\[
(\Theta, \rho, t) \xrightarrow{\lambda(A, s, d)} (\Theta, \rho \setminus \rho(A, s, d), t)
\]

where \(t < s\). That is to say, a computation which has already started in the system is not allowed to leave. We make the assumption \(t < s\) to simplify the model.

### B. Syntax and Semantics

The well formed formulas \(\psi\) are defined as follows:

\[
\psi ::= \text{true} \mid \text{false} \mid \text{satisfy}(\rho(\gamma, s, d)) \mid \\
\text{satisfy}(\rho(\Gamma, s, d)) \mid \text{satisfy}(\rho(\Lambda, s, d)) \mid \\
\neg \psi \mid \Diamond \psi
\]

A ROTA well formed formula can be an atomic proposition, which may be the value \(\text{true}, \text{false}, \) a “satisfy” function on a resource requirement \(\rho\), or a well formed formula with a logic operator “¬” (not), or temporal operators “\(\Diamond\)” (eventually) and “\(\Box\)” (always).

The semantics of ROTA is defined by the satisfaction symbol \(\models\), on a **computation path**, which is defined as follows.

**Definition 2: Computation Path** Let \(\chi \in S \times S\) be a binary relation such that \((S_i, S_j) \in \chi\) if there exists a transition rule \(S_i \xrightarrow{\xi} S_j\), \(S_i \xrightarrow{\xi} S_j\) where \(\xi\) is a resource located type. A

\[\Box\]

Here, “\(\Box\)” can be either an actor’s name, or empty
computation path is one branch of the tree frame that relation \( \chi \) on \( S \) produces.

Therefore, the tree structure that relation \( \chi \) on \( S \) produces represents all the possible evolutions of the system, and a computation path states one of the possible traces of the computation.

As shown in Figure 1, the ROTA semantics is defined by the satisfaction symbol \( \models \) on a computation path \( \sigma \) at time \( t \). We assume the system state that \( \sigma, t \) specifies is \( S = (\Theta, \rho, t, \gamma) \), and \( \Theta_{\text{expire}} \) gives the union of the resource sets which will expire during the time interval \((\max(s, t), d)\) according to path \( \sigma \). In other words, these are unwanted resource which will expire unless new computations requiring them enter the system. This creates opportunity for the system to accommodate new computations.

\[ M, \sigma, t \models \text{true} \]
\[ M, \sigma, t \models \text{false} \]
\[ M, \sigma, t \models \text{satisfy} \gamma(r, s, d) \]
\[ \text{if } f(\bigcup_{s \leq t < m-1} \Theta_{\text{expire}}, r, s, d) = \text{true} \]
\[ M, \sigma, t \models \text{satisfy} \gamma(\rho(\sigma, s, d)) \]
\[ \text{if } 3t_1, ..., t_{m-1}, \text{ such that } s \leq t_1 < ... < t_{m-1} < d, \]
\[ \rho(\Gamma, s, d) = \rho(\Gamma_1, \psi_1, \Gamma_2, \Gamma_3), \]
\[ M, \sigma, t \models \text{satisfy} \gamma(\rho(\sigma, s, d)) \]
\[ \text{where the state } M, \sigma', \psi \text{ specifies is } \Theta, \rho(\sigma, s, d), t \]
\[ M, \sigma, t \models \neg \psi \text{ if } M, \sigma, t \models \psi \]
\[ M, \sigma, t \models \psi \text{ if } \exists \sigma' \text{ such that } \forall' t > \sigma, M, \sigma, t' \models \psi \]
\[ M, \sigma, t \models \neg \psi \text{ if } \forall' t > \sigma, M, \sigma, t' \models \psi \]

**Theorem 3: Meet Deadline**

Suppose the state of the system is \( S_0 = (\Theta, 0, t), \) having \( \Theta \) resources but no computations to use them at time \( t \), the computation \( \Gamma \) can be completed by deadline \( d \), if and only if there exists a computation path \( \sigma \), denoted by \( (S_0', S_1', ..., S_n'), \) where \( S_0' = (\Theta, \rho(\Gamma, t, d), t), \) such that \( S_n' = (\Theta'_{\text{finish}}, t_n, t_n < d). \)

**Proof.** Assume we have a such a computation path \( \sigma \), at each of the time points when a subcomputation \( \Gamma_i \) is completed, we divide the path. So at the end we get \( m \) sub-paths, each of which represents a subcomputation of \( \Gamma, s, d. \) Apparently he time points we get, \( t_1, t_2, ..., t_n \) satisfy the complex resource requirement of \( \Gamma, s, d. \) according to Theorem 2, the computation can be completed by time \( d. \) This proves “if”.

Assume the computation \( \Gamma, s, d \) can be completed by the system. Since the system tree represents all possible evolutions of the system, there must be a path \( \sigma \), in which all actions in \( \Gamma \) are completed by time \( d. \) This proves “only if”. \( \square \)

The next question is whether a system can accommodate a new computation at a certain state, without affecting the existing computations in the system, as shown in Theorem 4, which answers the question raised in Section IV.

**Theorem 4: Accommodate Additional Computation**

A new computation \( \Gamma, s, d \) can be accommodated, without affecting the current executing computations in the system, if there exists a computation path \( \sigma \), such that resources which are expiring on \( \sigma \) during the time interval \((s, d)\), i.e., \( \bigcup_{s \leq t < m-1} \Theta_{\text{expire}}, \) satisfies the complex resource requirement of computation \( \Gamma, s, d. \)

**Proof.** Because resources \( \bigcup_{s \leq t < m-1} \Theta_{\text{expire}} \) on path \( \sigma \) satisfies the complex resource requirement of computation \( \Gamma, s, d. \), according to Theorem 3, we can build a path \( \sigma_T \) starting from state \( \bigcup_{s \leq t < m-1} \Theta_{\text{expire}}, \Gamma, s, d, s, \) which eventually reaches a state \( (\Theta', t_{\text{end}}, t_{\text{end}}) \) where \( t_{\text{end}} < d \). We then combine the two paths \( \sigma \) and \( \sigma_T \), in the way that we combine the transition rules of \( \sigma_T \) to the transition rules in \( \sigma \), which has the same start and end time, to form new concurrent transition rules. We call the new path generated by the combination \( \sigma' \), which is a path which accommodates computation \( \Gamma, s, d \), without affecting the existing computations which are already accommodated. According to Theorem 3, the computation \( \Gamma, s, d \) can be accommodated by the system, without affecting other computations. \( \square \)

Furthermore, ROTA allows reasoning about temporal properties of the system, such as “a computation can eventually be accommodated”, or “a computation can always be accommodated”, as illustrated in the semantics.

**VI. Conclusion**

Coordinating resource use among distributed computations is critical in open distributed systems. A key challenge is determining whether resources would be available to carry out an additional computation. In this paper, we presented ROTA, a resource-oriented temporal logic for distributed systems. Inspired by the CyberOrgs model, ROTA defines resources in time and space, and represents distributed computations in terms of the resources they require and the intervals during which they would like to use them. Given the state of resource availability at a point in time, ROTA tells us whether a particular deadline-constrained distributed computation can be accommodated by the system. This can be useful for computations choosing between various courses of action, allowing them to avoid attempting infeasible pursuits.
Work is ongoing in a number of directions. First, although ROTA does address reasoning about a wide range of distributed computations, in its current form, it does not address the wider range of actor computations where actors can interact. A particular challenge is to address the consequences of unpredictable delays in receiving messages or returning from blocking create requests on knowledge of when resources would be required in the future. It appears that it would be better to break down an actor’s computation into sequences of independent computations separated by states in which it is waiting to hear back from a blocking operation. We are currently looking into this possibility. Second, we are looking at allowing actors to migrate during the course of their execution. Particularly, an actor could continue to execute at its current location or migrate elsewhere, carry out part of its computation, and then return and resume. Comparing these choices presents some interesting challenges.

Finally, algorithmic complexity of the reasoning enabled by ROTA is obviously high. However, the context in which we hope to use ROTA is that of resource encapsulations of the type defined by the CyberOrgs model, where the reasoning only needs to concern itself with resources available inside the encapsulation. We are in the process of studying the extent to which using ROTA in the context of CyberOrgs ameliorates the complexity challenge. The prospect of tying resources required for reasoning with the size and complexity of the resource encapsulation being reasoned about for the purpose of empowering computations to choose encapsulation sizes is particularly attractive. In other words, if computations can determine the value of carrying out a computation, that can inform their decision about how much resource to expend in needed searching for resources before giving up.

VII. ACKNOWLEDGMENTS

This research is supported by a Discovery Grant from the Natural Science and Resource Council of Canada, a New Opportunities Fund Award from the Canadian Foundation for Innovation, and an Innovation and Science Fund Award from the Government of Saskatchewan. Their support is gratefully acknowledged.

REFERENCES


