Performance Analysis of Distributed Systems Involving Loops

Toqeer Israr
Eastern Illinois University
taisrar@eiu.edu

Abstract

This paper analyzes the performance of a distributed global system, composed of sub-services sequenced with strict and weak loop. We provide general formulas with which we calculate the performance of a composite service based on the performance of the constituent sub-services and the control operators. To do this, we model each sub-service as a Partially Ordered Specification (POS), where each sub-service is characterized by independent input and output events, and the minimum delays between these events. This technique allows two or more sub-services to be combined hierarchically. Proofs of correctness for these formulas are given with a discussion of an example throughout the paper.

Introduction

Many commercial systems, especially Cloud based applications, are generally the result of aggregation of various types of services (fine-grained / composite services). These are often structured as distributed systems, involving several communicating components running on different processors or in different processes. In the rapidly changing world of technology, no system is ever constant. These requests for changes, due to various reasons such as evolving customer requirements etc., have profound effects on the modeling and performance of a distributed global system, especially if the system is the result of aggregation of multiple sub-services involving several communicating components running on different processors or processes. For example, a system to analyze data of an online retail application could be implemented on a multi-tiered system, where the data is broken down and each sub-analysis is analyzed by several servers. The reults of these sub-analysis are then combined back to give an overview of customer's shopping pattern to enable the company to predict leading trends, customer habits, prepare for demands, optimize pricing and promotions, etc.

One needs to examine the performance of a distributed global system, especially when affected when either a sub-service or an involved component is replaced. A global system could be a system implemented using Cloud Technology working with a Big Data architecture. This becomes even more complex if we make the assumption that execution of all components do not start or end at the same time instant.

Focus of this Research

This paper focusses on three aspects of the performance of a global distributed system:

- First, we calculate the performance of a global service given the performance delays of the sub-services, especially when we assume not all involved components may start their executions at the same time or end at the same time.
- Secondly, we calculate how long a particular component is involved in a service, so we can infer when it becomes available for other services Note, a component may complete all its executions long before all the executions of a service and hence an involved component completion time will always be less than or equal to the completion time of a service in which the component is involved in.
- Thirdly, we examine the affects on performance of the global service by modification of the existing global service – either a sub-service, involved in the global service, is replaced by a another sub-service or a component, involved in the global service, is replaced with another component with different performance parameters.

Israr previously conducted research focused on a global service composed of multiple sub-services that were sequenced with strict and weak sequencing, concurrent and alternative sequencing operators" [4,5]. The research conducted for this paper focused on analyzing performance of the global services, where the global service is composed of sub-services sequenced with strict and weak while loops. In addition, we propose our analysis can be extended to analyze performance of parts of Big Data framework.

We start off with discussing an example of an online retail system using Big Data framework. We review the previous work done in this domain, and then we present our research.

Big Data Analysis

An online retailer with thousands of online customers can serve as an example of Big Data analysis. As number of transactions increase (sometimes up to 500,000 transactions / second – see [2]), so does the data being generated, causing companies somtimes to implement Big Data strategies in a Cloud based environment. This typically involves tools and technologies such as Hadoop, Pig, Hive, etc. to collect, interpret, and analyze data from many angles with the goal of discovering a previously hidden insight, which in turn can provide a competitive advantage or address a pressing business problem.

Figure 1.0 illustrates a MapReduce infrastructure of a Hadoop eco system and the generated data from an online retail transactions is labelled as "Big Data". MapReduce is a framework for processing parallelizable problems across huge datasets using a large number of servers (components). A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. The outputs from the reduce function yield the desired end results.

In this case, the data processing done by *map* functions could be determining trends for the products purchased. Each *map()* function, each involved two servers, are analyzing data from different geographical locations such that *map()*$_1$, *map()*$_2$ and *map()*$_3$ are analyzing data from transactions originated from North America, Europe and rest of the world respectively.

The reduce functions are bringing all the analysis back together to enable the company to predict leading trends, customer habits, prepare for demands, optimize pricing and promotions, etc. These map and reduce functions can be modeled as activities (sub-services) which could be sequenced using a sequence, a choice, and/or loop operators.

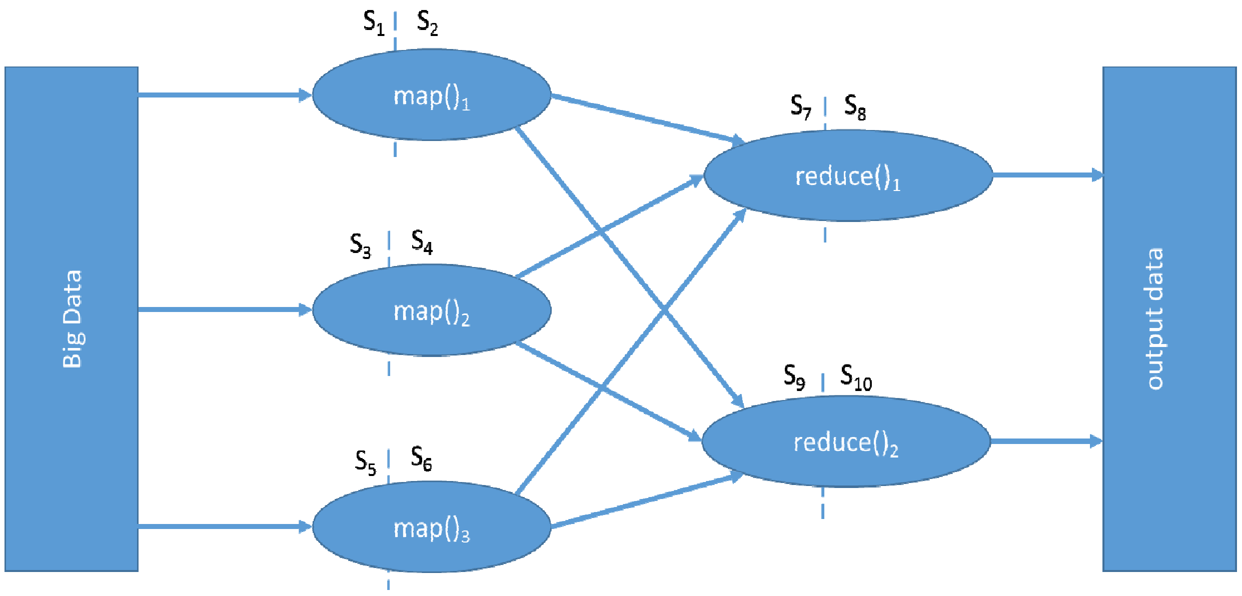


Figure 1.0 – Map Reduce Architecture

As mentioned, no state of a system is ever constant and is always evolving. From the performance point of view, one questions what is the affect on the performance of the complete data analysis (end-to-end) if i) the nature of map/reduce jobs change (a service change) or ii) the number of servers or the servers (S_1, S_2, S_3…) (component change) processing these jobs are replaced. Also, when do these servers become available for other services. As, each of these *map()* and *reduce()* functions involved multiple servers, one needs to realize also that not all the servers will always be ready to process the request, and could cause a time delay.

Modeling

Numerous methodologies have been defined to model distributed systems. Most, if not all, of notations can potentially decompose a system into *sub-systems* and further into *sub-sub-systems*, where the final decomposition typically identifies a single component [1]. However, quite often in many distributed systems, even the final decomposition involves multiple components. This is often referred as the *crosscutting* nature of distributed services [1], i.e. services involve several collaborating components playing different roles that each component may participate in several services. By modeling according to the roles, the behaviour of the components can be defined precisely, but the behaviour of a service is fragmented. To model the global behaviour of a service explicitly, an orthogonal view is needed where the collaborative behaviour is the focus [1]. Within this orthogonal view, there is an inherent need to show the relationships and dependencies amongst the involved roles, especially when all the involved roles may potentially start at different times and end at different times.

Modeling Distributed Activities

To satisfy such needs, based on UML Activities [9], Israr et al. in [4] and [5], propose a new modeling paradigm represented as a partially ordered set of inputs and outputs, called Partial Ordered Specification (POS) as shown in Figure 2. This modeling paradigm models independent starting and ending events that may occur at different times. The POS paradigms illustrates the dependencies between these events and can be used in analyzing performance of activities, with various sequencing operators.

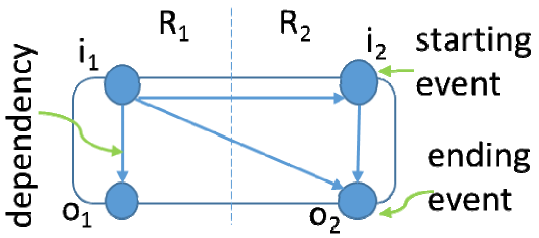


Figure 2 – Partial Order Specification

Israr et al. suggests for a given activity, the input and the output of each involved role can be modeled by a starting and an ending event as shown by filled-in circles in Figure 2 [4]. These events, belonging to various components, form a partially ordered set, where a causal relationship may exists between some of these events, indicated by arrows in Figure 2. The ending events are not ordered relative to one another, but each ending event has a dependency on its corresponding starting event (local sequencing).

Figure 2 illustrates an activity, with starting events i_1 and i_2 and ending events o_1 and o_2. As i_1 and o_1 are events of the same role R_1, o_1 must occur after i_1 due to local sequencing. Furthermore, since i_2 has a dependency on i_1, all the events in the activity, must occur after i_1. Due to the relationship $i_1 \rightarrow i_2$ and $i_2 \rightarrow o_2$, there is another dependency from i_1 to o_2.

Strict and Weak Sequencing

Two activities C1 and C2 are said to be strongly sequenced when all the sub-activities of C1 are completed before any sub-activity of C2 starts. However, weak sequencing between C1 and C2 means that each role locally applies sequencing to the local sub-collaborations of C1 and C2. This implies that a role may start with sub-collaborations that belong to C2 as soon as it has completed all its local sub-collaborations that are part of C1. Strong sequencing implies weak sequencing, but not inversely. In weak sequencing, if a role is not involved in C1, it may start with sub-collaborations of C2 even before C1 begins its execution.

Performance

Performance of such systems at a bigger scale can be very difficult to analyze. Chuang et al. discussed a stochastic Petri-net workflow model to propose various performance formulas for basic routing pattern of a workflow system [11]. Li et al. extended the Workflow net (WF-net) with timing information to provide a formal framework for modeling and performance analysis [7]. In their work, they proposed a method for computing the lower bound of the average turnaround time of transaction instances in a given workflow. Similar work has been done by Hao et al. [3], Wang et al. [11], and Lazowska et al. [6]. McNeile [8] modeled and analyzed end-to-end workflow delays but their analysis assumed that all the components are available at the beginning of the workflow, yielding a single workflow delay.

All of the above mentioned work assumes a workflow implemented as an execution of activities implemented by a single role. With a workflow involving multiple roles, the very first concern is the starting and ending time of each role involved in the given workflow – time of the starting and ending event of each role. Secondly, within a given collaboration, dependencies need to be identified which may exist between various events of different roles. Based on these dependencies, then performance can be analyzed for not only events involved in local sequencing but also between events of one role and events of another or same role. As such we shall explore well-structured collaborations with multiple roles, sequenced with standard UML operators to yield a global collaboration, describing an abstract service.

Delay for a Given Activity

Israr et al. introduced an approach to determine the dependencies amongst the input and output events within a given sub-activity [4]. For a given sub-activity, according to this approach, one measures the delay between the time instance of the occurrence of input event i and a dependent output event o, provided all the other events on which o depends have occurred long time before. This delay is called Nominal Execution Time Delay (NETD), written as Δ^i_o. This led to the following formula which yields the performance of a collaboration D based on dependencies between input and output events:

$$_DT_o = \max_{i \varepsilon I(D)} ({}_DT_i + {}_{(cp)D}\Delta^i_o) \tag{1}$$

where T_o is the time of the output event o, T_i is the time of the input event i, Δ^i_o is the NETD from input event i to an output event o and I(D) is the set of input events. Subscript “D” indicates all the notations are for the abstract activity D and *(cp)* depicts a control flow path, a single execution of a given system depicting a single control flow (SCF) as compared to multiple executions of a system i.e. multiple control flows (MCF). If the events are independent, then the NETD between an input event i and a non-dependent output event o’ is assumed to be:

$$_D\Delta^i_{o'} = -\infty \tag{2}$$

Hence, the formula in (1) is not limited to dependent events, but rather is applicable for all involved events – **dependent and independent events** alike.

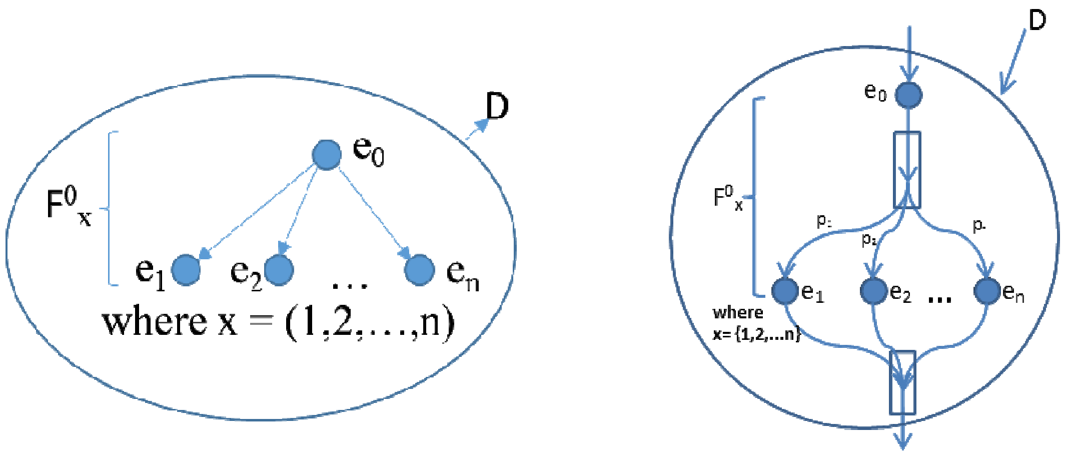
Shared Resources: Israr et al. assumed the NETD is attained during a control flow path, which may not be realistic if shared resources are involved in the processing of several inputs on which a single output depends on [4]. Hence the assumption is made that there are no shared resources and each role or all concurrent activities of a given role are implemented by an independent processor.

Types of Delays

For the modeling of the performance of collaborations, we consider that the NETD that capture the performance of a collaboration could be of one three types: fixed delays, range of delays, and delays defined in terms of probability distributions – stochastic delays. For Stochastic Delays (SD), written as ${}_{(stoc)}{}^{(cp)}{}_{D}\Delta^{i}{}_{o}$, we assume that the execution delays (measured or specified) are of stochastic nature and are defined by a probability distribution that could be measured by performing a large number of delay measurements. We recognize that stochastic delays could follow any kind of distribution. Hence, we provide general formulas that can be applied to any kind of distribution. A Special Case of Stochastic Delay is the Fixed Delay (FD). If a given NETD has a stochastic delay where the distribution is a Dirac Delta function, then we can say that the NETD has a determinist duration or a fixed delay (FD) ${}_{(fixed)}{}^{(cp)}{}_{D}\Delta^{i}{}_{o}$. This means that the delay for a given control flow path cp always results in the same value provided the starting conditions (or initial states) are the same for all participating components. Fixed delays are commonly used when there is a need to specify performance for a single control flow path for hard real time control systems. For example, a performance requirement could be that a traffic light takes exactly 45 seconds to go from green to red, exactly another 20 seconds to go from green to yellow and exactly another 2 seconds to go from yellow to red. We say that a NETD ${}^{(cp)}{}_{D}\Delta^{i}{}_{o,}$ has a range of delays (RD) where the delay specified for a control flow cp may provide different values, and it is important to specify minimum and maximum values, written as ${}_{(max)}{}^{(cp)}{}_{D}\Delta^{i}{}_{o}$ and ${}_{(min)}{}^{(cp)}{}_{D}\Delta^{i}{}_{o}$ respectively. In reality, the delays may be of stochastic nature, but one is normally not interested in the precise form of the probability distribution. Range of delays is used to describe the behavior of system models for hard real-time systems, using for instance the formalization of Timed Automata. If the range is infinitesimally small, where the ${}_{(max)}{}^{(cp)}{}_{D}\Delta^{i}{}_{o} - {}_{(min)}{}^{(cp)}{}_{D}\Delta^{i}{}_{o} \approx 0$, then this results in a Fixed Delay as detailed above.

Basic Performance Characteristics of an Activity

Let us consider the various orderings of events e_0, e_1, ...e_n in Figure 3 and abstract the sequence of events $e_0, e_1,\ldots,e_n$ by a composite activity D, where composite activity D abstracts concurrent events in Figure 3.



a – concurrency b - alternative

Figure 3 – Sequencing Operators

We assume there exists a stochastic delay, $F^w{}_x(t)$, between events e_w and e_x as shown in Figure 3, characterized by a cumulative distribution function (CDF) $F^w{}_x(t)$. We assume these delays are statistically independent. If the distributions of the delays between the events are considered to be a delta distribution, then these delays have a deterministic duration, and we have the situation of "Fixed Delays". In the following, we examine fixed delays and stochastic delays for the abstract activity D for the concurrent operator.

Concurrency - Stochastic Delays:

If we are interested in the time it would take for the earliest event amongst $e_1, e_2,\dots e_n$ after event e_0 to occur as shown in Figure 3, then the earliest or "minimum" CDF of a global activity D composed of parallel events e_x, can be modeled by [10] as:

Completion of the Earliest Event: $${}_{\min}F_D(t) = 1 - \prod_{x=1}^{n}(1 - F^0{}_x(t)) \qquad (3)$$

Next, the completion of the global activity D requires all of the events e_x to occur. So we need to find the delay for the occurrence of the last event e_x. This can be accomplished by calculating the delay between events with the maximum time delay [10]:

Completion of the Last Event: $${}_{\max}F_D(t) = \prod_{x=1}^{n} F_x^0(t) \qquad (4)$$

Concurrency - Fixed and Range of Delays

If we consider the delays between the events as a delta distribution, and if we consider a range of delays for the deterministic duration of the abstract activity D, then the range of delays can be defined as:

Completion of the Earliest Event: $${}_{\min}F_D = \min_{x=1\dots n}(F_x^0) \qquad (5)$$

Completion of the Last Event: $_{\max}F_D = \max_{x=1\ldots n} (F_x^0)$ (6)

These formulae can be used to determine the distribution of a global scenario given the distribution function of between individual events.

Alternatives

Alternative execution is when there is a decision to be made amongst multiple successor events and only one of the successor events occurs as shown in Figure 3b.

Stochastic Delays:

To obtain a distribution delay, each path is assigned a probability value p_i leading to alternative paths i. If we examine the alternative case, such as in Figure 3b, then the overall distribution of D is defined by [10]:

$F_D(t) = \sum_{i=1}^{n} p_i F_i^0(t)$ (7)
where p_i is the probability for event e_i to occur.

Fixed Delays:

The fixed delay of the abstract activity D is the delay of the particular single control flow path selected to be executed:

$F_D = F_i^0$ (8)
where e_i is the event occurs and $1 \leq i \leq n$

If we consider the range of delays, then we need to consider all the control flow paths which would lead to the greatest and smallest delay:

Minimum delay: $_{\min}F_D = \min_{i=1\ldots n}(F_i^0)$ (9)
Maximum delay: $_{\max}F_D = \max_{i=1\ldots n}(F_i^0)$ (10)

These formulae can be used to determine the distribution of a global scenario given the distribution function of between individual events.

Deriving General Formulas for Various Operators

In the following, we consider multiple iterations of collaboration A with strict and weak while loops, forming a well-structured composite collaboration D, followed by collaboration B (we do not model collaboration B as that has been discussed in [4]).

We consider NETDs to be fixed, a range or stochastic delays. This is annotated by $_{(mzz)D}\Delta^w_z$ for the delay from the starting event w ε I(D) to the ending event z ε O(D), where *mzz* is the delay type, i.e. *mzz = fixed, max, min* or *stoc.* The delay $_{(mzz)D}\Delta^w_z$ of the composite collaboration D depends on the participation of the roles w and z in the sub-collaborations A and/or B, *w* represents the role of the input and *z* represents the role of the output in collaboration D. As such, for each operator, the formulas are classified according to this participation. We define R(*X*) as the roles involved in collaboration *X*, I(*X*) and O(*X*) as all the input and output events respectively of collaboration *X*.

Israr et al. proposed definitions of Δ^i_o for strict, weak, concurrent and alternative sequencing operators [4]. In the following, we extend this work by analyzing performance of sub-activity A and B sequenced with **weak** and **strict while loop** operators with **independent input** and **output** events. With these definitions, one can calculate the completion time of the global activity and the total time a componenet is involved in a sub-activity as well as a global activity [4]. These compositions are abstracted by activity D.

Strict While Loop

Figure 4a shows a control flow diagram of a strict while loop repeating sub-collaboration A. We do not show the follow-up collaboration "B" as the analysis for a sub-collaboration strictly sequenced with B has been previously analyzed in [4]. Similar to definitions of strick and weak sequencing, Bochmann defines a strict while loop where collaboration C_1 is repeated and then followed by C_2, as: "C_1 is executed zero, one or more times and then C_2 will be executed; more precisely, the behavior starts with a choice between C_1 and C_2; if C_1 is executed, there is **strict** sequencing between the end of C_1 and the choice of executing C_1 again or terminating the loop with C_2.", written as "C_1 *s C_2" [1]. Before each iteration of C_1, a set of roles C* (represented by r_{c*}… to $r_{c*'}$ in Figure 4b), where C* ε R(D), makes a choice for the execution of C1 repeatedly or to terminate the loop and execute C_2, where all the roles in C* make the same choice. No action in either sub-collaboration A or B may start to execute until this choice is made. We model this by adding a sub-collaboration "Choice" preceding the sub-collaborations C_1. Choice sub-collaboration has roles R(Choice) = C* ∪ R(A) ∪ R(B), where we introduce dependencies from the starting events of C* to the ending events of R(Choice), which ensures none of the roles start their execution in C_1 or C_2 until the choice is complete. We assume the act of making the choice and the propagation of this choice to R(Choice) (shown by dependency arcs in Choice sub-collaboration in Fig 4b) is done instantly, and does not add any delay.

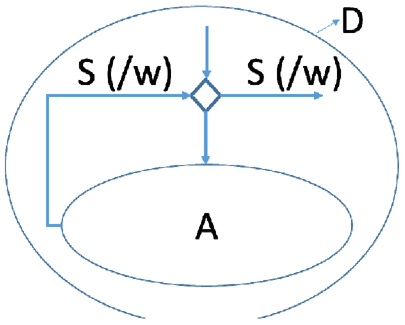


Figure 4a – strict (/weak) while loop

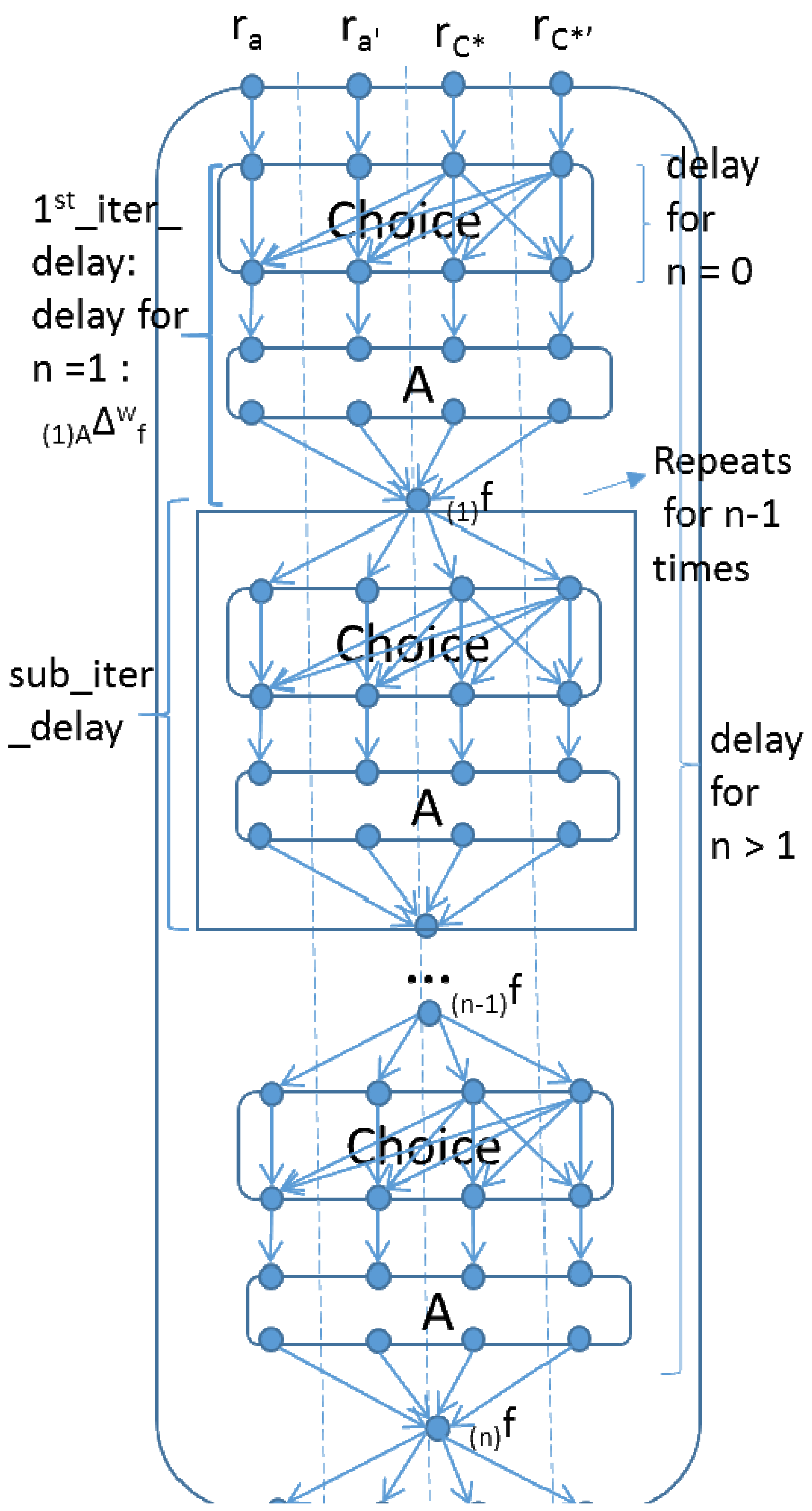


Figure 4b – POS of Strict While Loop

Figure 4b shows a partial order diagram that defines the dynamic behaviour of the strict while loop for the control flow path where sub-collaboration A is executed n times. The special case of n=1 is also indicated. In case of n=0, the delays are zero. As this is a strict while loop, all the ending events of each iteration of collaboration A synchronize at a synchronization event, written as $_{(j)}f$ for iteration j. To keep track of the iteration number, we introduce the notation $_{(j)}\alpha$, where α represents any of the behavioural properties of a collaboration and j is an integer representing the number of the iteration. The index j is shown for only those item that relate to different iterations. However, there is no explicit synchronization before the starting events of the first iteration of collaboration *Choice* and A, that is, we do not make any assumption about the kind of sequence that precedes the composite collaboration D.

Consideration of a Single Control Flow Path

If we consider a single control flow path of a strict while loop where sub-collaboration A repeats n times, the NETD of the composite collaboration D $^{(cp)}{}_{D}\Delta^{w}{}_{z}$ for w ε I(D) and z ε O(D) is given by the formulas in Table 1.

Table 1 – Strict While Loop Operator (Single Control Flow)

n		**Fixed Delays / Range of Delays**	
0	if w ε C*	0	(11a)
	Otherwise	$-\infty$	(11b)
1	if w ε C*	$\max(_{y\,\varepsilon\,I(A),\ x\,\varepsilon\,O(A)}\,({}_{(1)}{}^{(cp)}{}_{(mzz)A}\Delta^{y}{}_{x}))$	(12a)
	Otherwise	$\max_{x\,\varepsilon\,O(A)}\,({}_{(1)}{}^{(cp)}{}_{(mzz)A}\Delta^{w}{}_{x})$	(12b)
> 1	if w ε C*	$\sum_{j=1}^{n}\ (\max_{y\varepsilon\,I(A),\ x\,\varepsilon\,O(A)}\,({}_{(j)}{}^{(cp)}{}_{(mzz)A}\Delta^{y}{}_{x}))$	(13a)
	Otherwise	$\max_{x\,\varepsilon\,O(A)}\,({}_{(1)}{}^{(cp)}{}_{(mzz)A}\Delta^{w}{}_{x})\ +$ $\sum_{j=2}^{n}(\max_{y\varepsilon\,I(A),\ x\varepsilon O(A)}\ {}_{(j)}{}^{(cp)}{}_{A}\Delta^{y}{}_{x})$	(13b)

In the following, we justify the formulas given in Table 1. We assume that all the dependencies shown in Figure 4b by an arrow are associated with a zero delay.

Case n = 0 – A executes 0 times, but Choice executes once

if w ε C*: All the involved roles cannot start their execution until the choice is made by roles $r_{c*}...r_{c*'}$ of collaboration Choice, causing a dependency from role $r_{c*}...r_{c*'}$ to all the involved roles, R(D). We had initially mentioned that we assumed that the delay to make the choice and for choice propagation is zero, and hence the delay from the roles $r_{c*}...r_{c*'}$ to all the involved roles is zero.

otherwise: As there is no execution of A, there is no dependency and hence no delay from any starting event to any ending event other than mentioned above. This is represented as –∞ as per defined by Equation (2).

Case n = 1 – A executes once, but Choice executes twice:

if w ∉ C*: Collaboration A executes once, which is equivalent to sub-collaboration A strictly sequenced with a following collaboration. As all the paths of execution merge at $_{(1)}f$, using (6), this delay is calculated as the maximum delay over all the outputs of A from the input w:

$$1^{st}\_iter\_delay = \max_{x\ \varepsilon\ O(A)} ({}_{(1)}{}^{(cp)}{}_{(mzz)A}\Delta^{w}{}_{x}) \qquad (14)$$

if w ε C*: If the starting event of role w belongs to a role making the choice, then no execution in collaboration A could have started until this starting event from role *w* occurs. Furthermore, the ending event of role z cannot occur until all the dependencies have been satisfied from all the starting events. Hence, using (6), the collaboration's delay is the maximum over all the starting and ending events as stated in Equation (12a).

Case n > 1 – Sub-collaboration A executes n times, and Choice executes n+1 times

From (12a) and (12b), we know the delay for A's first iteration.

As seen in Figure 4b, the *subsequent_iter_delay* looks identical to the *1st_iter_delay* with the addition of the synchronization event *f* at the beginning of each iteration, with the dependencies. When we calculated the delay for 1st_iter_delay above, we assumed that starting events except starting event of role *w* have occurred some time ago. Hence we do not consider the delays from remaining starting events to the synchronization event $_{(1)}f$. This is not the case for the 2nd iteration and onwards, as the starting events for those iterations cannot occur until $_{(n-1)}f$ has occurred. Only then, the execution of the nth iteration may start. Since the delay from all starting events need to be considered, applying (6) to (14), we get:

$$subsequent\_iter\_delay = \max_{w\ \varepsilon\ I(A)} (\max_{x\ \varepsilon\ O(A)} ({}_{(j)}{}^{(cp)}{}_{A}\Delta^{w}{}_{x})) \qquad (15)$$

For each of these iterations, the NETD depends on the execution path of the body of A being executed. Hence, this delay could be different for each iteration depending on the control flow path in A. The total delay can be calculated as:

$$2\_to\_n\_delay = \sum_{j=1}^{n}(\max_{w\ \varepsilon\ I(A)} (\max_{x\ \varepsilon\ O(A)} ({}_{(j)}{}^{(cp)}{}_{A}\Delta^{w}{}_{x}))) \qquad (16)$$

From Figure 4b, it is quite evident that the NETD of the composite collaboration is the sum of *1st_iter_delay* and *2_to_n_delay*, which yields (13b).

Consideration of Multiple Control Flow Path – Range of Delays

If we consider all the possible control flow paths, then based on the formulas from Table 1, it is quite evident that the minimum delay occurs for n = 0 and the maximum delay occurs when n = ∞. This leads to:

Minimum Delay: ${}^{(cp)}{}_{(min)D}\Delta^{w}{}_{z} = 0 \quad \text{if } w\ \varepsilon\ C^*$ (17a)

$^{(cp)}{}_{(min)D}\Delta^{w}{}_{z} = -\infty$ otherwise (17b)

Maximum Delay: $^{(cp)}{}_{(max)D}\Delta^{w}{}_{z} = \infty$ (18)

Consideration of Multiple Control Flow Paths – Stochastic Delays

For stochastic delays, it is assumed that each time the Choice is performed, there is a probability p that A is executed and a probability *q=1-p* to stop the iterations. Then the NETD for collaboration D, $^{(cp)}{}_{(stoc)D}\Delta^{w}{}_{z}$, is given by:

if w ε C*: $q * \sum_{i=1}^{n} p^{i} \otimes_{j=1}^{i} {}_{(j)}{}^{(cp)}{}_{(stoc)A}\Delta^{w}{}_{z}$ (19)

Otherwise: $pq * \prod_{x\, \varepsilon\, O(A)} {}_{(1)}{}^{(cp)}{}_{(stoc)A}\Delta^{w}{}_{x}(t) \otimes$ (20)

$(\delta + \sum_{i=1}^{n-1} p^{i} \otimes_{j=1}^{i} \prod_{y\, \varepsilon\, I(A), x\, \varepsilon\, O(A)} {}_{(j)}{}^{(cp)}{}_{(stoc)A}\Delta^{w}{}_{x}(t))$

and there is the probability q that there is no dependency from w to z, when w ∉ C*. This happens when n = 0.

The above delay is calculated by applying Equation (7) to the formulas in Table 1. We apply probability p for each time A executes and probability q once for A to stop the iteration and we sum up these delays.

<u>if w ε C*</u>: If the starting event belongs to the roles involved in making the Choice, then applying (7) to (11a), (12a) and (13a) and summing up the delays yields (19).

<u>Otherwise</u>: If the starting event does not belong to a role involved in making the Choice, then applying (7) to (11b), (12b) and (13b) and summing up the delays yields (20). If the sub-collaboration does not iterate at all (i.e. n = 0), then there obviously is no delay from w to z and hence no dependency from w to z.

Weak While Loop

Israr et al. defines a weak while loop with body C_1 followed by collaboration C_2 similar to that of a strict while loop "...except that **weak** sequencing is used between the end of C_1 and the choice of executing C_1 again or terminating the loop with C_2." This is annotated as C1 *w C2 [1]. Figure 5 shows a POS definition of such a weak while loop.

Consideration of a Single Control Flow – Fixed Delays

NETD: If we consider a single control flow path of a weak while loop where sub-collaboration A repeats n times, the NETD for w ε I(A) and z ε O(A) for the composite collaboration D, $^{(cp)}{}_{D}\Delta^{w}{}_{z}$, is given in Table 2.

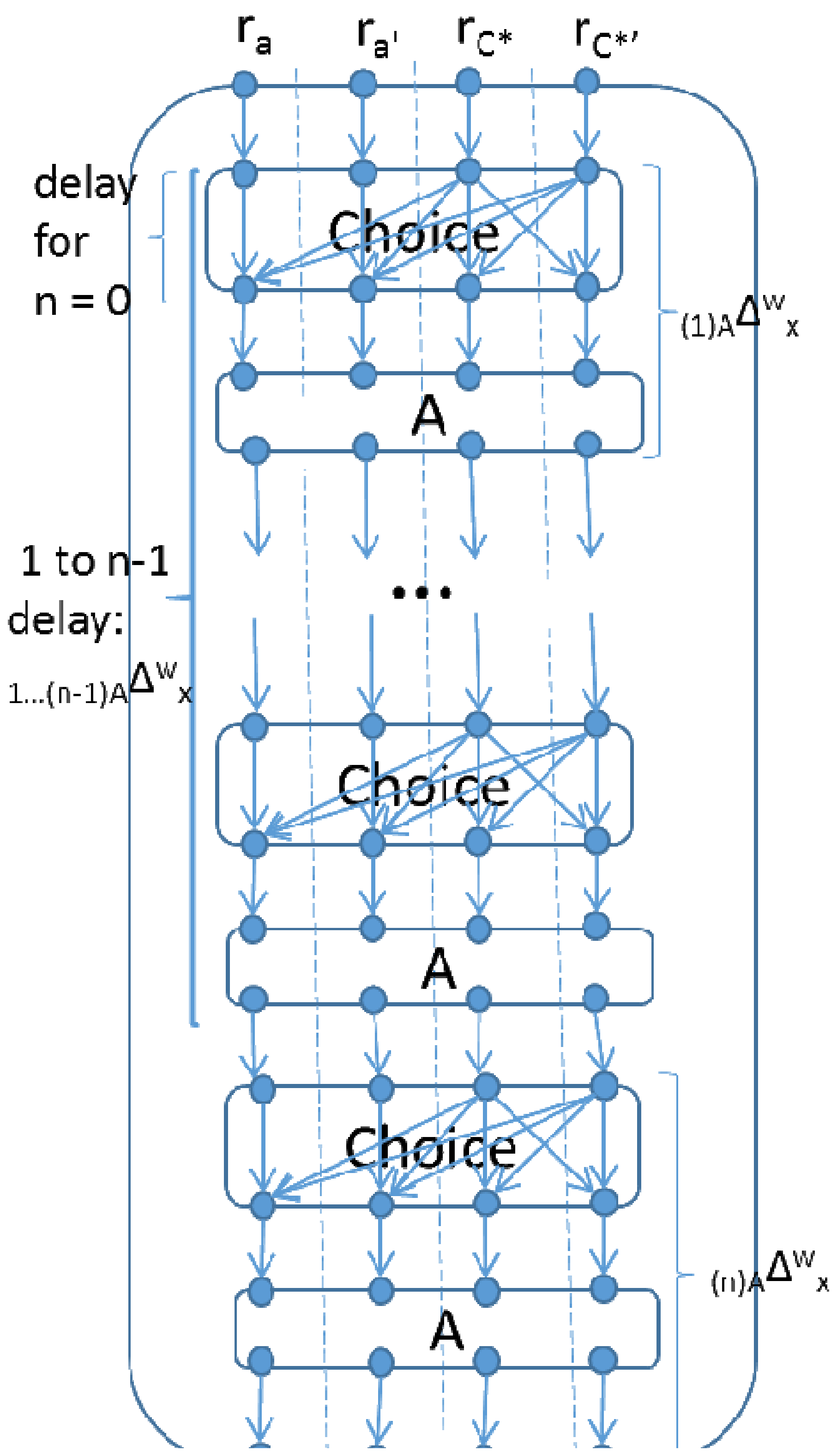


Figure 5 – POS of Weak While Loop

Table 2 – Weak While Loop

n		**Fixed Delays / Range of Delays**	
0	if $w \varepsilon C^*$	0	(21a)
	Otherwise	$-\infty$	(22b)
1	if $w \varepsilon C^*$	$\max_{y\varepsilon I(A)} ({}_{(1)}{}^{(cp)}{}_{(mzz)\,A}\Delta^{y}{}_{z})$	(23a)
	Otherwise	${}_{(1)}{}^{(cp)}{}_{(mzz)A}\Delta^{w}{}_{z}$	(23b)
> 1		$\max_{x\varepsilon O(A)} (\ {}_{1\ldots(n-1)}{}^{(cp)}{}_{(mzz)\,D}\Delta^{w}{}_{x} + {}_{(n)}{}^{(cp)}{}_{(mzz)A}\Delta^{x}{}_{z})$	(24)

Case n = 0: The explanation is the same as case n = 0 for strict while loop as in section 3.1.1.

Case n =1: The proof for (23a) and (23b) is similar to the proof of Equations (12a) and (12b) of Table 1. However, none of the roles may start their execution in A until the choice is made by the roles in C*. For the ending event of z to occur, dependencies from all the starting events of A to the ending event of z must occur, and no action in A may start until the choice is made by all roles in Choice collaboration. If w belongs to C*, then dependencies from the starting events of all the roles to the ending event of z must be satisfied and therefore we consider delays from the starting events of all the roles to the ending event of the desired role in A.

If w does not belong to C*, the only dependency remaining is from the starting event of w to the ending event of z and hence we only consider the delay between these two events.

Case n > 1 : This is calculated using a recursive formula where ${}_{(n)}{}^{(cp)}{}_{D}\Delta^{w}{}_{z}$ denotes the NETD for the nth iteration of the composite collaboration D.

As can be seen in Figure 5, the 1 to (n-1) iterations of collaboration A can be abstracted by ${}_{1\ldots(n-1)}{}^{(cp)}{}_{D}\Delta^{w}{}_{x}$. To calculate the nth iteration of collaboration A, we consider first the previous (n-1) iterations with the delay ${}_{1\ldots(n-1)}{}^{(cp)}{}_{D}\Delta^{w}{}_{x}$ which are weakly sequenced with the nth iteration with the delays ${}_{(n)}{}^{(cp)}{}_{D}\Delta^{w}{}_{x}$. The formula for weak sequencing of these delays is discussed in the work presented in [4].

Consideration of Multiple Control Flow Paths – Stochastic Delays

We consider all the possible control flow paths and assume that each leads with probability p to the execution of A and probability *q=(1-p)* to the termination of the loop. Based on the formulas for fixed delays, the NETD for collaboration D, ${}^{(cp)}{}_{(stoc)D}\Delta^{w}{}_{z}$ can be calculated as:

$$q \sum_{i=1}^{n} p^{i} * {}_{(i)}{}^{(cp)}{}_{(stoc)D}\Delta^{w}{}_{z} \qquad (28)$$

and there is the probability q that there is no dependency from w to z, when $w \notin C^*$. This happens when n = 0. The proof is very similar to the proof of (19) and (20). If the role w does not belong to C*, there exists a probability q for no dependency from w to z. This can happen when n=0.

Conclusion

We use Israr et al.'s [4] method of representation to model collaborations and analyze various scenarios. We have considered the delays of composite activities sequenced with strict and weak while loops and have calculated the delays for such composite activities as well as for the individual components involved. We believe that this approach to the performance modeling of distributed system designs is useful in many fields of application, including performance analysis of Cloud computing, Big Data as well as distributed workflow management systems, e-commerce applications, and/or Web Services.

We have implemented a tool that takes as input an Activity Diagram with defined performance characteristics and provides outputs as the NETDs of the global collaboration for fixed delays. Even though, our work is quite mathematical and proofs were provided throughout the paper, we still would like to apply this research to an Industrial Case Study such as the MapReduce example discussed earlier to illustrate the application of our work.

References

[1] Bochmann, G.v., "Deriving component designs from global requirements", in Proceedings of International Workshop on Model Based Architecting and Construction of Embedded Systems (ACES), Toulouse, 2008, pp 55-69.

[2] Garcia, Ahiza. "Amazon 'Prime Day' Shattered Global Sales Records". CNNMoney. N.p., 2016. Web. 17 Apr. 2016.

[3] Hao, J., Pei-an W., "An approach for workflow performance evaluation based on discrete stochastic Petri net." e-Business Engineering, 2007. ICEBE 2007. IEEE International Conference on. IEEE, 2007.

[4] Israr, Bochmann, Performance Modeling of Distributed Activity Services, Proceedings of the 2nd ACM/SPEC International Conference on Performance Engineering, Karlsruhe, Germany, 2011, pp 475-480

[5] Israr T., Bochmann G.v., Stochastic performance analysis of distributed activities, Proc. of 5th Intern. Workshop on Non-functional Properties in Modeling: Analysis, Languages and Processes co-located with 16th Intern. Conf. on Model Driven Engineering Languages and Systems, Miami, USA, 2013

[6] Lazowska E., Zahorjan J., Graham G., Sevcik K., Quantitative system performance: computer system analysis using queuing network models, Prentice-Hall, Upper Saddle River, NJ, 1984

[7] Li J., Fan Y., and Zhou M. "Performance modeling and analysis of workflow." In Proceedings of IEEE Transitions of Systems, Man and Cybernetics, 229-242, 2004

[8] McNeile, A., "Using Motivation and Choreography to model Distributed Workflow" Proceedings of the 5th ACM SIGCHI Annual International Workshop on Behaviour Modelling- Foundations and Applications, Montpellier, France, 2013

[9] Object Management Group. Unified Modeling Language Superstructure, V2.1.2. OMG Available Specification, November 2007. OMG document number: ptc/2007-11-02.
[10] Sahner R., Trivedi K., Performance and reliability analysis using directed acyclic graphs. IEEE Trans. Software Eng., pp. 1105–1114, 1987
[11] Wang Y., Lin C., Ungsunan P. Huang X., "Modeling and survivability analysis of service composition using Stochastic Petri Nets." The Journal of Supercomputing, 79-105, 2011

Biography

TOQEER ISRAR is currently an Assistant Professor at Eastern Illinois University. He earned his B.Eng. and MASc in Electrical Engineering degree from Carleton University, ON (Canada); and, a Ph.D. in Electrical and Computer Engineering, 2014, from University of Ottawa, ON (Canada). He is an internationally recognized expert in the areas of performance and software engineering and has over 5 years of experience in the Industry. He is a registered Professional Engineer in Ontario (Canada). Dr. Israr may be reached at taisrar@eiu.edu.