Performance Analysis of Distributed Systems Involving Loops

Toqeer Israr, Eastern Illinois University

Abstract

In this study, the author evaluated the execution and performance of a distributed global system (global activity) comprised of sub-services sequenced with strict and weak loops. For this global activity, performance was determined based on the performance and sequence types of the constituent sub-activities. This was achieved by modeling each -activity as a partially ordered specification (POS), such that each sub-activity was identified by independent input and output events and the minimum delays between these events. This technique allowed hierarchical composition of two or more sub-activities.

Introduction

Many commercial systems, especially cloud-based applications, are the result of an aggregation of various types of services (fine-grained/composite services), and will be referred to as activities in this paper. Quite often, these composite activities may involve multiple interacting components located on different processors. 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, have significant effects on the performance of such systems. For example, a system for analyzing the data of an online retail application could be implemented on a multi-tiered system, where the data are broken down for several servers to analyze each sub-analysis. The results of these sub-analyses are then combined to give an overview of a customer's shopping pattern in order to enable the company to predict leading trends and customer habits, prepare for demands, and optimize pricing and promotions.

For this current study, the performance of a distributed global system was examined, in particular for when either a sub-service or an involved component changes. A global system could be a system implemented using cloud technology working with big data architecture. This becomes even more complex as the assumption is made that execution of all components does not start or end at the same instant.

Focus of this Research

This study focused on the following:

- First, the performance of a composite activity was determined, given the performance delays of its subsystems; this becomes quite interesting when a particular assumption is made such that not all involved components may start or end their executions at the same time.
- Second, how long a particular component is involved in a service was calculated, such that an inference can be made about its availability for other services Note that a component may complete all of its executions long before all of 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.
- Third, the effects of the global service on performance were examined 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.

In previous studies by Israr [1] and Israr and Bochmann [2], the authors focused on a global service composed of multiple sub-services that were sequenced with strict and weak sequencing, concurrent and alternative sequencing operators. This current study was 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, the analyses in this study can be extended to analyze performance of parts of a big data framework.

Big Data Analysis

An online retailer with thousands of online customers can serve as an example of a big data analysis. As the number of transactions increase (sometimes up to 500,000 transactions per second) [3], so does the data being generated, causing companies occasionally to implement big data strategies in a cloud-based environment. This typically involves tools and technologies such as Hadoop, Pig, and Hive 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 illustrates a MapReduce infrastructure of a Hadoop eco system and the generated data from an online retail transaction labeled as "big data." MapReduce was used to process parallelizable problems across enormous datasets using a great number of servers (components). According to the tutorial, "A MapReduce job typically splits the input data-set into independent sub-data-sets, which are analyzed and processed by the map tasks in a completely parallel manner. This framework then sorts the outputs of the maps" [4] for use as inputs to reduce the number of tasks. The outputs from the reduce function yield the desired end results.

In this case, the data processing done by the *map* functions could be determining trends for the products purchased. Each *map()* function involves two servers and analyzes data from different geographical locations, such that *map()*$_1$, *map()*$_2$, and *map()*$_3$ are analyzing data from transactions originating from North America, Europe, and rest of the world, respectively. The reduce functions bring all of the analyses back together to enable the company to predict things such as leading trends, customer habits, prepare for demands, optimize pricing, and promotions. These map and reduce functions can be modeled as activities (sub-services) that can be sequenced using a sequence, a choice, and/or loop operators.

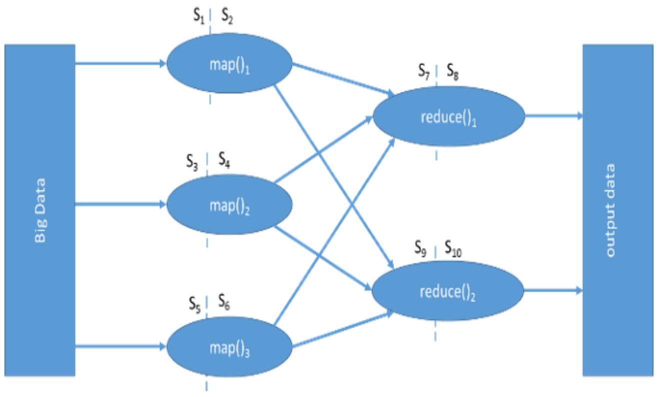


Figure 1. MapReduce Architecture

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

Modeling

Several methodologies have been used to describe the modeling of a distributed composite system. Typically, several of these models can possibly be refined into sub-activities and additionally into sub-sub-activities, where the end result is often identified with a single component [5]. However, that is not necessarily practical in many distributed systems, as even the final decomposition involves multiple components. This is also known as the "crosscutting" nature of distributed services identified with multiple collaborating components, where each component contributes to more than one service. The behavior of the components can be specified precisely how they are modeled by their roles. However, the behavior of the service becomes fragmented. Another view is required, where the behavior is focused on modeling the global behavior of a given service explicitly [5]. In this view, there would be an essential need to demonstrate the relationships and the dependencies among the roles (implemented on various components) that are involved. This becomes essential when the roles do not necessarily start and end their execution at the same time.

Modeling Distributed Activities

To satisfy such needs, partial order specification (POS), a new modeling paradigm, was introduced. Figure 2 shows how this new modeling paradigm, POS, models a partially ordered set of inputs and outputs [1, 2], which enables the events occurring at different times to be modeled. These events are typically starting and ending events, which can be independent. Using these events, POS illustrates the dependencies between different actions of a role (such as ending of a role and starting of another role) and allows one to model a service as an activity and analyze its performance.

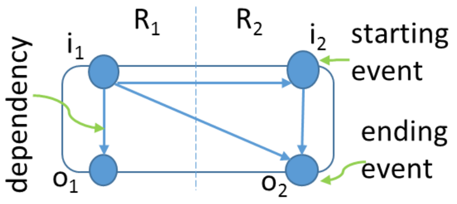


Figure 2. Partial Order Specification

For a given role, a POS models a starting and an ending event, as a filled-in circle in Figure 2 [1]. As can be seen, a partially ordered set can be formed of these events, giving rise to causal relationships between some of the events—these relationships are depicted by arrows in Figure 2. A starting event marks the beginning of the execution of the first action for a given role, while an ending event marks the ending of the execution of the last event for a given role in a given activity. Even though the ending events may not be ordered relative to each other, there exists a causal relationship between each starting event and an ending event of the same role, also known as local sequencing. Figure 2 depicts an activity having two starting events (i_1 and i_2) and two ending events (o_1 and o_2). For events of the same role, the ending event must occur after the starting event of the same role, due to local sequencing (i.e., o_1 occurs after i_1 and o_2 occurs after i_2). Additionally, it can be seen that i_1 causes i_2 to occur. This, in turn, along with local sequencing, postulates that all of the events in the activity must occur after i_1. Due to the relationships $i_1 \rightarrow i_2$ and $i_2 \rightarrow o_2$, there is another dependency from i_1 to o_2.

Strict and Weak Sequencing

Strong sequencing between two activities, for example where A2 occurs after A1, implies that all of the actions of all of the roles in A1 must have executed to completion before the action of any role in A2 may begin. In contrast, weak sequencing between the same two activities, A1 and A2, implies that there exists a local sequencing between A1 and A2 such that a role may start executing actions of A2 as soon as that role has completed execution of all its actions in A1. Whenever strong sequencing exists, weak sequencing exists as well, but not inversely. Furthermore, in weak sequencing, a role may start execution of its actions in A2 if it is not involved in A1 and, hence, execution of A2 may start before execution of A1 even begins.

Performance

Performance of such systems on a larger scale can be very difficult to analyze. Huang et al. [6] discussed a stochastic Petri-net workflow model to propose various performance equations for basic routing pattern of a workflow system. Li et al. [7] extended the Workflow net (WF-net) with timing information to provide a formal framework for modeling and performance analysis. 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 Wang et al. [6], Hao and Pei -an [8], and Lazowska et al. [9]. McNeile [10] modeled and analyzed end-to-end workflow delays, but that analysis assumed that all of the components were available at the beginning of the workflow, thereby yielding a single workflow delay.

All of the aforementioned work assumed that a workflow implemented as an execution of activities was implemented by a single role. With a workflow involving multiple roles, the very first concern are the starting and ending times of each role involved in the given workflow—times of the starting and ending events 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, performance can be analyzed for not only events involved in local sequencing but also between events of one role and events of another or similar role. As such, well-structured collaborations with multiple roles, sequenced with standard UML operators to yield a global collaboration describing an abstract service, were explored.

Delay for a Given Activity

Israr [1] introduced an approach for determining the dependencies among the input and output events within a given sub-activity. For a given sub-activity, according to this approach, the delay is measured between the time instance of the occurrence of input event i and dependent output event o, provided all of the other events on which o depends have occurred a long time before. This delay is called nominal execution time delay (NETD), written as Δ^i_o. This led to Equation (1), which yields the performance of a collaboration, D, based on dependencies between input and output events:

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

where, T_o is the time of output event o; T_i is the time of input event i; Δ^i_o is the NETD from input event i to output event o; and, $I(D)$ is the set of input events. Subscript "D" indicates that all of 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).

If the events are independent, then the NETD between input event i and non-dependent output event o' is assumed to be determined by Equation (2):

$$_D\Delta^i_{o'} = -\infty \qquad (2)$$

hence, Equation (1) is not limited to dependent events, but rather is applicable for all involved events—dependent and independent events alike.

Israr [1] assumed that 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. Hence, the assumption was made that there were no shared resources, and each role or all concurrent activities of a given role would be implemented by an independent processor.

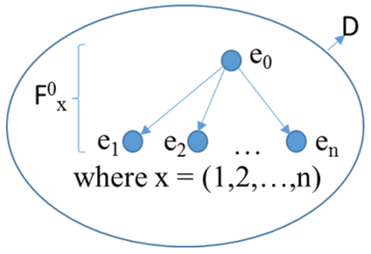
Types of Delays

For the modeling of the performance of collaborations, the NETD was considered, such that capture of the performance of a collaboration could be one of three types: fixed delays, range of delays, and delays defined in terms of probability distributions—stochastic delays. For stochastic delays (SD), written as $_{(stoc)}{}^{(cp)}{}_{D}\Delta^{i}_{o}$, it was assumed that the execution delays (measured or specified) were of a stochastic nature and were defined by a probability distribution that could be measured by performing a large number of delay measurements. Stochastic delays could follow any kind of distribution. Hence, general equations are provided 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 it can be said that the NETD has a determinist duration, or a fixed delay (FD), of $_{(fixed)}{}^{(cp)}{}_{D}\Delta^{i}_{o}$. 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.

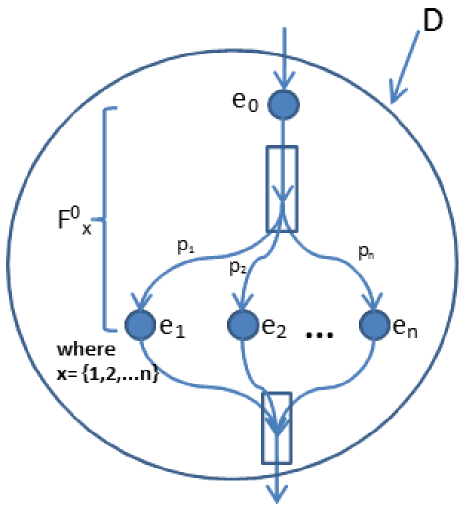
For example, a performance requirement could be that a traffic light takes exactly 45 seconds to go from green to red, exactly another 20 seconds to go from green to yellow, and exactly another 2 seconds to go from yellow to red. An NETD $^{(cp)}{}_{D}\Delta^{i}_{o}$ can have a range of delays (RD), 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 actuality, the delays may be of a 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 $_{(max)}{}^{(cp)}{}_{D}\Delta^{i}_{o} - {}_{(min)}{}^{(cp)}{}_{D}\Delta^{i}_{o} \approx 0$, 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. It was assumed that there existed a stochastic delay, $F^{w}_{x}(t)$, between events e_w and e_x, as shown in Figure 3 and characterized by a cumulative distribution function (CDF), $F^{w}_{x}(t)$. It was further assumed that these delays were statistically independent. If the distributions of the delays between the events were considered to be a delta distribution, then these delays would have a deterministic duration, leading to the situation of "Fixed Delays."



(a) Concurrency



(b) Alternative

Figure 3. Sequencing Operators

Concurrency: Stochastic Delays

To determine the time it would take for the earliest event among $e_1, e_2,\ldots e_n$ after event e_0 to occur, as shown in Figure 3, then the earliest or "minimum" CDF of global activity D, composed of parallel events e_x, can be calculated using Equation (3), which represents the completion of the earliest event [11]:

$$_{\min}F_D(t) = 1 - \prod_{x=1}^{n}\left(1 - F_x^{0}(t)\right) \tag{3}$$

Next, the completion of global activity D requires all of the events e_x to occur and, hence, the delay for the occurrence of the last event e_x needs to be calculated. This can be accomplished by calculating the delay between events with the maximum time delay [11], as defined in Equation (4), which represents the completion of the last event:

$$_{\max}F_D(t) = \prod_{x=1}^{n} F_x^0(t) \tag{4}$$

Concurrency: Fixed and Range of Delays

If the delays between the events are assumed to be a delta distribution, and if a range of delays are considered for the deterministic duration of the abstract activity D, then the range of delays can be defined by Equations (5) and (6), which represent the completion of the earliest and last events, respectively:

$$_{\min}F_D = \min_{x=1\ldots n}\left(F_x^0\right) \tag{5}$$

$$_{\max}F_D = \max_{x=1\ldots n}\left(F_x^0\right) \tag{6}$$

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

Alternatives: Stochastic and Fixed Delays

Alternative execution occurs when there is a decision to be made among multiple successive events, and only one of the successor events occurs, as shown in Figure 3(b). To obtain a distribution delay, each path is assigned a probability value p_i, leading to alternative paths i. If the alternative case is examined, such as in Figure 3(b), then the overall distribution of D is defined by Equation (7) [11]:

$$F_D(t) = \sum_{i=1}^{n} p_i F_i^0(t) \tag{7}$$

where, p_i is the probability for event e_i to occur.

The fixed delay of the abstract activity D is the delay of the particular single control flow path selected to be executed as defined in Equation (8):

$$F_D = F_i^0 \tag{8}$$

where, e_i is the event occurring and where $1 \leq i \leq n$.

If the range of delays is considered, then all of the control flow paths need to be considered, which would lead to the greatest and smallest delays, as calculated by Equations (9) and (10) for the minimum and maximum delays, respectively:

$$_{\min}F_D = \min_{i=1\ldots n}\left(F_i^0\right) \tag{9}$$

$$_{\max}F_D = \max_{i=1\ldots n}\left(F_i^0\right) \quad 10)$$

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

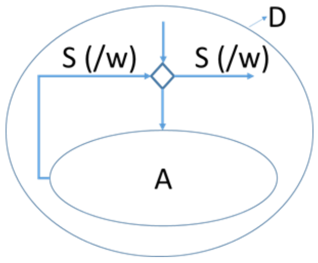
Deriving General Equations for Various Operators

For this current study, NETDs were considered to be fixed, range, or stochastic delays. This was annotated by $_{(mzz)\mathrm{D}}\Delta^{w}{}_{z}$ for the delay from the starting event, w ε I(D), to the ending event z ε O(D), where *mzz* was the delay type (i.e., mzz = fixed, max, min or stoc). The delay $_{(mzz)\mathrm{D}}\Delta^{w}{}_{z}$ 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 equations were classified according to this participation. R(*X*) describes the roles involved in collaboration *X*, while I(*X*) and O(*X*) describe all of the input and output events of collaboration *X*, respectively.

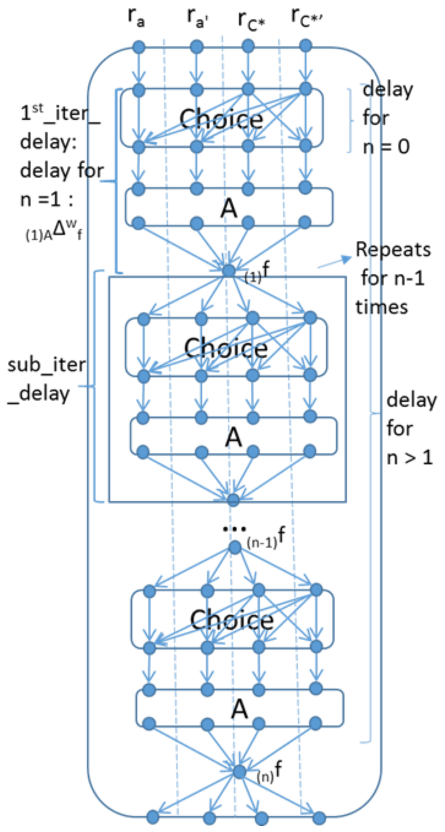
Israr [1] proposed definitions of $\Delta^{i}{}_{o}$ for strict, weak, concurrent, and alternative sequencing operators. In the following section, these were extended by analyzing performance of sub-activity A and B sequenced with **weak** and **strict while loop** operators with **independent input** and **output** events. With these definitions, one can calculate the completion time of the global activity and the total time a component is involved in a sub-activity as well as a global activity [1]. These compositions are abstracted by activity D.

Strict While Loop

Figure 4(a) shows a control flow diagram of a strict while loop repeating sub-collaboration A. The follow-up collaboration "B" is not modeled (or analyzed), as the analysis for a sub-collaboration strictly sequenced with B was previously analyzed [1]. Similar to definitions of strict and weak sequencing, Bochmann [5] 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." Before each iteration of C_1, a set of roles C* [represented by r_{c*}… to r_{c*}, in Figure 4(b)], 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 of 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. This is modeled 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 dependencies are introduced 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. It is assumed that 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 4(b)] is done instantly, and does not add any delay.



(a) Strict (/weak) While Loop



(b) POS of Strict While Loop

Figure 4. While Loops

Figure 4(b) shows a partial order diagram that defines the dynamic behavior 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 of 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, the notation $_{(j)}\alpha$ is introduced, where α represents any of the behavioral 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, no assumption is made about the kind of sequence that precedes the composite collaboration D.

Consideration of a Single Control Flow Path

If a single control flow path of a strict while loop is considered, 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 Equations (11)-(13) 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	$\max_{x\ \varepsilon\ O(A)}\ (_{(1)}{}^{(cp)}{}_{(mzz)A}\Delta^{w}{}_{x}) +$ (13b) $\sum_{j=2}^{n} (\max_{y\varepsilon\ I(A),\ x\varepsilon O(A)}\ {}_{(j)}{}^{(cp)}{}_{A}\Delta^{y}{}_{x})$

For Equations (11)-(13), it was assumed that all of the dependencies shown in Figure 4(b) by an arrow are associated with a zero delay.

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

If w ε C*, none of the involved roles can 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 of the involved roles, R(D). It is assumed that the delay to make the choice and for choice propagation to be zero, and, hence, the delay from the roles $r_{c*}...r_{c*'}$ to all of 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 those mentioned previously. This is represented as $-\infty$, as per Equation (2).

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

If $w \notin C^*$, collaboration A executes once, which is equivalent to sub-collaboration A strictly sequenced with a following collaboration. As all of the paths of execution merge at ${}_{(1)}f$, using Equation (6), this delay was calculated as the maximum delay over all of the outputs of A from input w, as defined by Equation (14):

$$\mathit{1^{st}\_iter\_delay} = \max_{x\,\varepsilon\,O(A)} \left({}_{(1)}{}^{(cp)}{}_{(mzz)A}\Delta^{w}_{x} \right) \tag{14}$$

If $w\ \varepsilon\ C^*$, and 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 of the dependencies have been satisfied from all of the starting events. Hence, using Equation (6), the collaboration's delay is the maximum over all of 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 Equations (12a) and (12b), the delay for A's first iteration is known. As seen in Figure 4(b), 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 the delay for 1st_iter_delay is calculated, it is assumed that starting events except starting event of role *w* have occurred some time ago. Hence, the delays from remaining starting events to the synchronization event, ${}_{(1)}f$, are not considered. 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 can the execution of the nth iteration start. Since the delay from all starting events needs to be considered, applying Equations (6)-(14) yields the subsequent iteration delay, as defined by Equation (15):

$$\mathit{subsequent\_iter\_delay} = \max_{w\,\varepsilon\,I(A)} \left(\max_{x\,\varepsilon\,O(A)} \left({}_{(j)}{}^{(cp)}{}_{A}\Delta^{w}_{x} \right) \right) \tag{15}$$

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, f, or each iteration, depending on the control flow path in A. The total delay can be calculated using Equation (16):

$$\mathit{2\_to\_n\_delay} = \sum_{j=2}^{n} \left(\max_{w\,\varepsilon\,I(A)} \left(\max_{x\,\varepsilon\,O(A)} \left({}_{(j)}{}^{(cp)}{}_{A}\Delta^{w}_{x} \right) \right) \right) \tag{16}$$

From Figure 4(b), 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 Equation (13b).

Consideration of Multiple Control Flow Paths: Range of Delays

If all of the possible control flow paths are considered then, based on the equations from Table 1, it is quite evident that the minimum delay occurs for n = 0 and the maximum delay occurs when $n = \infty$, which gives rise to Equations (17a/b) and (18), minimum and maximum delay, respectively:

if $w\ \varepsilon\ C^*$,

$${}^{(cp)}{}_{(\min)D}\Delta^{w}_{z} = 0 \tag{17a}$$

otherwise,

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

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

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 Equations (19) and (20):

if $w\ \varepsilon\ C^*$,

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

otherwise,

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

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

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 above delay is calculated by applying Equation (7) to the equations in Table 1. The probability p can be applied for each time A executes and probability q once for A to stop the iteration and delays are summed. If $w\ \varepsilon\ C^*$, then the starting event belongs to the roles involved in making the Choice, then applying Equations (7)-(11a), (12a), and (13a) and summing the delays yields Equation (19).

Otherwise, the starting event does not belong to a role involved in making the Choice, then applying Equations (7) -(11b), (12b), and (13b) and summing the delays yields Equation (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 [1] 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 [5]. Figure 5 shows a POS definition of such a weak while loop.

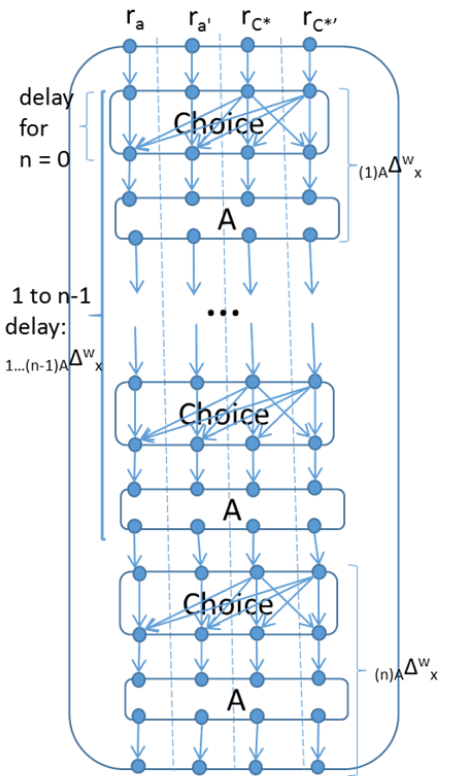


Figure 5. POS of the Weak While Loop

Consideration of a Single Control Flow: Fixed Delays

If a single control flow path is considered of a weak while loop, where sub-collaboration A repeats n times, equations to calculate NETDs for w ε I(A) and z ε O(A) for the composite collaboration D, $^{(cp)}{}_{D}\Delta^{w}{}_{z}$, are given by Equations (21)-(24) in Table 2.

Table 2. Weak While Loop

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

For case n = 0, justification is the same as for case n = 0 for the strict while loop. For case n =1, the proof for Equations (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 of 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, delays from the starting events of all the roles to the ending event of the desired role in A are considered. 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, only the delays between these two events are considered.

Case n > 1 is calculated using a recursive equation, where ${}_{(n)}{}^{(cp)}{}_{D}\Delta^{w}{}_{z}$ denotes the NETD for the nth iteration of the composite collaboration D. Figure 5 shows 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, first the previous (n-1) iterations with the delay ${}_{1\ldots(n-1)}{}^{(cp)}{}_{D}\Delta^{w}{}_{x}$ are considered, which are weakly sequenced with the nth iteration with the delays ${}_{(n)}{}^{(cp)}{}_{D}\Delta^{w}{}_{x}$. The equation for weak sequencing of these delays is discussed in the study by Israr [1].

Consideration of Multiple Control Flow Paths: Stochastic Delays

In this case, all of the possible control flow paths are considered 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 equations for fixed delays, the NETD for collaboration D, $^{(cp)}{}_{(stoc)D}\Delta^{w}{}_{z}$ can be calculated by Equation (25):

$$q \sum_{i=1}^{n} p^{i} * {}_{(i)}{}^{(cp)}{}_{(stoc)D}\Delta^{w}_{z} \qquad (25)$$

where, 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 Equations (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.

Conclusions

Israr's [1] method of representation was used to model collaborations and analyze various scenarios. The delays of composite activities sequenced with strict and weak while loops were considered, and the delays were calculated for such composite activities as well as for the individual components involved. This approach to the performance modeling of distributed system designs can be 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. Also, a tool was implemented 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 this work was quite mathematical and proofs were provided throughout the paper, it would beneficial to illustrate this research with an industrial case study such as the MapReduce example discussed earlier.

References

[1] Israr, T. (2011) Performance modeling of distributed activity services. *Proceedings of the 2nd ACM/SPEC International Conference on Performance Engineering,* (pp. 475-480). Karlsruhe, Germany.

[2] Israr, T., & Bochmann, G. V. (2013). Stochastic Performance Analysis of Distributed Activities. *Proceedings of the 5th International Workshop on Non-Functional Properties in Modeling: Analysis, Languages and Processes*, (24-31). Miami, FL.

[3] Garcia, A. (2015, July 16). Amazon "Prime Day" shattered global sales records. *CNNMoney*. Retrieved from http://money.cnn.com/2015/07/15/news/amazon-walmart-sales/

[4] MapReduce tutorial. (2013, August 4). Retrieved from https://hadoop.apache.org/ docs/r1.2.1/mapred_tutorial.html

[5] Bochmann, G. V. (2008). Deriving component designs from global requirements. *Proceedings of International Workshop on Model Based Architecting and Construction of Embedded Systems,* (pp. 55-69). Toulouse, France.

[6] Wang, Y., Lin, C., Ungsunan, P., & Huang, X. (2011). Modeling and survivability analysis of service composition using stochastic Petri nets. *The Journal of Supercomputing*, *56*(1), 79-105.

[7] Li, J., Fan, Y., & Zhou, M. (2004). Performance modeling and analysis of workflow. *IEEE Transactions on Systems, Man and Cybernetics—Part A*, *34* (2), 229-242.

[8] Hao, J., & Pei-an W. (2007). An approach for workflow performance evaluation based on discrete stochastic Petri net. *Proceedings of the IEEE International Conference on e-Business Engineering*. Hong Kong, China.

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

[10] McNeile, A. (2013). Using Motivation and Choreography to Model Distributed Workflow. *Proceedings of the 5th ACM SIGCHI Annual International Workshop on Behavior Modelling—Foundations and Applications*, (1-11). Montpellier, France.

[11] Sahner, R., & Trivedi, K. (1987). Performance and reliability analysis using directed acyclic graphs. *IEEE Transactions on Software Engineering*, (10), 1105–1114.

Biography

TOQEER ISRAR is an assistant professor at Eastern Illinois University. He earned his BEng and MASc in Electrical Engineering degrees from Carleton University, Ontario, Canada, and a PhD in Electrical and Computer Engineering, 2014, from the University of Ottawa, Ontario, Canada. He is an internationally recognized expert in the areas of performance and software engineering and has over five years of experience in industry. He is a registered professional engineer in Ontario, Canada. Dr. Israr may be reached at taisrar@eiu.edu