Superlinear Relative Speedup of the Service Layered Utility Maximization Model for Dynamic Webservice Composition in Virtual Organizations

Abiud Wakhanu Mulongo
School of Computing and Informatics
University of Nairobi
Nairobi, Kenya
Email: abiudwere [AT] gmail.com

Elisha Abade
School of Computing and Informatics
University of Nairobi
Nairobi,Kenya

William Okello Odongo
School of Computing and Informatics
University of Nairobi
Nairobi, Kenya

Elisha T. O. Opiyo
School of Computing and Informatics
University of Nairobi
Nairobi, Kenya

Bernard Manderick
Artificial Intelligence Lab
Vrije Universiteit Brussel
Brussels, Belgium

Abstract— **Dynamic webservice composition (DWSC) is a promising technology in supporting Virtual organizations to autogenerate composite services that maximize the utility of Internet commerce service consumers over a range of the consumer's QoS constraints. However, over the last decade DWSC remains a non polynomial deterministic problem, making its industrial application limited. Local planning approaches to the problem guarantee solutions in polynomial time but lack capabilities to solve for inter workflow task constraints that could be critical to a service consumer. The inability of local planning algorithms to capture global constraints means that the techniques are likely to yield low quality solutions. Among the existing global planning approaches, mixed integer programming (MIP) has been found to be the best tradeoff in guaranteeing global quality of solutions albeit with the possibility of solutions being generated in exponential time. This makes MIP techniques limited for small scale problems. In [23] we proposed SLUM, a technique that combines the relative advantages of local planning and MIP to achieve solutions that are more efficient but 5% less quality than MIP, but 5% more quality than the local planning approach [25]. However, it remains unknown whether or not SLUM could be more efficient than the standard MIP (S-MIP) in the absence of service elimination in layer 1 of the optimization process, leading to the question, *can SLUM exhibit superlinear speedup relative to S-MIP*? Using formal mathematical analysis, this study established that SLUM can be more efficient up to a maximum of 2^{k-1} times than S-MIP, where *k* is the number of sequential workflow tasks. Further, using experimentation with differential calculus and empirical relative complexity coefficients for analysis , the study established that it would take 3 years to achieve a superlinear speedup of 2^{k-1} times when *k=2* and a speedup of 1.5 times in 28 hours. The conclusion is that even in the absence of webservice elimination, virtual enterprise brokers can still benefit from the relative efficiency gains of SLUM up to a practical limit of 50% over S-**

Keywords--- *Web Service Composition, Virtual Organizations, Decomposition, Superlinear, Optimization, Mixed Integer Programming, Service Layered Utility Maximization, Empirical Complexity*

I. INTRODUCTION

Webservices technology is a pillar of modern Internet based Business to Business (B2B) and Business to Customer (C2B) interactions [9]. Rabelo et al [26] identifies webservices and the concept of *webservice composition* as essential components of the ICT infrastructure framework for agile virtual collaborative networked organizations. A web service is a distributed software component that enables machine to machine interaction over the network using standard network protocols such as the Simple Object Access Protocol[1] and *REST* . In ICT-enabled VOs, webservices are the software components that produce the data required to execute one of the business tasks required to fulfill a particular business process e.g an online purchase order process. Webservice composition on the other hand, is a process that involves the discovery, selection, linking and execution of a set of atomic distributed webservices in a specified logical sequence in order to service complex customer request that none of the services could fulfill singularly [2][3][4]. By making use of a webservice composition middleware, a virtual enterprise broker, in response to a complex consumer need, can quickly generate a more value added composite service from

already existing individual webservices that are owned by geographically dispersed virtual enterprises. Because different functionally similar individual webservices can exhibit varying Quality of Service (QoS) from time to time, the need for dynamic webservice composition strategies that are able to generate a composite service that maximizes a service consumer's utility over a range of the consumer's QoS preferences and constraints is critical [9]. This need is more pronounced in VOs , since customer centricity and business agility are key differentiating factors of Virtual Organizations as detailed in [1-3], [20 -21], [26]. In addition, dynamic webservice composition can increase the chances of the service composition process surviving faults during execution through preplanning mechanisms such as the one proposed in [9] and [Dustdar], thereby boosting service reliability as perceived by service requestors.

A. The Dynamic Webservice Composition Problem

In spite of its potential benefits to virtual enterprise brokers, dynamic composite webservice selection remains a nondeterministic polynomial (NP) hard problem [7], [10-12] over the last 15 years, making its applicability severely constrained. The coupling of several challenges account for the difficulty. Firstly, the need to satisfy various local and global QoS constraints that vary from consumer to consumer and from time to time - QoS attributes as evidenced by the papers in [6-9]. Secondly, availability of many different functionally similar webservices though with different QoS – the need to isolate the best set of services that form the composite service given the QoS needs. Third, the complexity and size of the business process that is executed by the webservices further increasing the computational effort required to select the best composite service. Fourth, the volatility of the service environment such that the execution of the composite service is prone to failure- there is need for fault tolerant and failure recovery enabled service composition strategies. The fourth issue has been addressed adequately in papers like [5], [22]. Even so, fault handling strategies when embedded within a service composition process are bound to increase the computational overhead [9]. This paper takes for granted issues to do with composite service fault handling. The other three issues present the webservice composition problem as an optimization problem of how to select the best quality composite service that satisfy a user's QoS constraints within a reasonable amount of time.

B. Local Planning and Mixed Integer Programming Strategies

Two well-known classes of optimization algorithms that tackle the problem are the local planning strategy and the Mixed Integer Programming initially formulated by Benatallah et al [9]. The local optimization method, herein L-MIP has been analytically and empirically proven to guarantee solutions within polynomial time [9], [25]. However, L-MIP lacks support for constraints that span workflow tasks and thus denies the user an opportunity to express important inter task QoS constraints such as " the composite service should cost more than a certain amount of money". Additionally, local planning strategies are known to be suboptimal and their solutions are about 20%-30% less quality in terms of weighted aggregate utility than the global planning MIP methods. Global planning method based on Mixed Integer Programming hereafter, SMIP, is able to consider both local and global constraints, guarantees globally optimal solutions but suffers exponential state explosion where one or a combination of the three variables is large enough: number of QoS constraints, number of candidate webservices per task or number of workflow tasks. Particularly Benatallah et al [2004] experimentally demonstrates that the runtime of S-MIP sores beyond 40 webservices per workflow task. More recently, Abiud et al [25] confirmed that S-MIP has an empirical superpolynomial runtime growth. In Abiud et al [25]. Thus the need for dynamic webservice composition methods that can are able to produce composite services of acceptable quality more efficiently is evident.

C. The Servoce Layered Utility Maximization Model

In [24], we proposed a two layered MIP technique for composite service selection, SLUM for Service Layered Utility Maximization. SLUM borrows the ideas of Layering as optimization decomposition as described in [14-17]. Like the SIMP method, SLUM is able to take into account both global and local constraints. SLUM starts with the entire set of webservices in stage one and solves a MIP subproblem using a subset of the webservice QoS constraints. In layer two, a second MIP subproblem is solved on the remaining subset of QoS constraints. In [25], the authors show that if t_{eA} is the running time of S-MIP on a problem instance and t_{eB} is the running time of SLUM on the same problem instance, $t_{eB} = 1.3t_{Ea}^{0.8}$. This means that for instance if the running time of S-MIP is 300 seconds, the running time of SLUM is $(300^{0.8} * 1.3) =$ 125 seconds which is about 2.4 times faster than S-MIP. The relative speedup of SLUM over S-MIP could be attributed to the fact that when using SLUM some candidate webservices are eliminated in stage one and only a few get into the second round of optimization, whereas in the second stage, the reduced solution space is searched against a subset of QoS constraints. The general argument is that using SLUM, the solution space is much reduced through early elimination of some candidate services [24].

A natural question that follows is ,*Can SLUM be more efficient than S-MIP when no candidate service is eliminated in stage one?* Superficially, the simple answer would be no. The reason for the answer could be that when no service is eliminated at stage one, SLUM would do the same amount of work as S-MIP and therefore would be at least equally efficient as S-MIP if not worse; worse because, SLUM as shown in [25] the performance of SLUM is also limited by the sequential overhead involved in moving data from one stage to another. However, the answer to the question is more complicated than this. Contrary to intuition, this study hypothesizes that SLUM is still more efficient than S-MIP even without elimination of candidate webservices. Our hypothesis is informed by the theory that decomposition of computational problems, even when performed sequentially, is more efficient than solving the original problem due to the fact that the complexity of problem grows more than linearly (superlinearly) [13]. Here we term the speedup of SLUM w.r.t S-MIP arising solely from the superlinearity property (without the effect of service elimination) of sequentially decomposed problems as *superlinear relative speedup* or simply *superlinear speedup*. Under the hypothesis, the study aimed to answer the research questions that follow in section 1.3.

D. *Research Questions*

The overall research question this paper addresses is *"How does the speedup of SLUM on a set of composite service selection problem instances of increasing hardness change relative to S-MIP in the absence of webservice elimination*?" This question is recast into the following research questions:-

RQ1: What is the maximum superlinear speedup of SLUM w.r.t S-MIP on a workflow with a known number of sequential tasks?

RQ2: Given a workflow with a known number of sequential tasks, what is the minimum problem size in the number of webservices per task that can benefit from superlinear speedup of SLUM?

RQ3: Given a workflow with a known number of sequential tasks, how much time would it take to achieve a SLUM superlinear speedup of a given magnitude for a given number of webservices per workflow task?

II. RELATED WORK

The authors in [24] qualitatively detail the superlinear property of the SLUM model for webservice composition. However, neither a formal nor empirical quantification of the superlinear behaviour of SLUM is provided. In Abiud et al [25], the relative speedup of SLUM is compared with that of S-MIP and the result is that S-MIP is initially 1.3 faster than SLUM on problem instances with less than 22 webservices per workflow task. At the same time, the authors also show that beyond 22 webservices, SLUM is much faster than S-MIP having an empirical relative complexity coefficient of about 0.8. Even though partly, superlinearity and partly the effect of pruning the initial solution space at layer 1 could account for the results, it's unknown how the scaling characteristics of SLUM would change when QoS constraints are adjusted such that no elimination of services at layer 1 happens i.e the question, can SLUM still yield composite solutions faster than S-MIP when the solution space remains the same across the two layers, remains unanswered. The work in this paper fills the gap and is dedicated to investigating the relative performance of SLUM w.r.t S-MIP without relying on service elimination.

III. MATHEMATICAL ANALYSIS

A combination of mathematical analysis and experimental evaluation are used. Mathematical analysis is first used to answer research question one, followed by experimentation that was used to answer research question two and three.

A. *Notations*

Hereafter, we adopt the notations below.

- q_1 : The number of QoS attributes at the Service Consumer Utility Maximization Layer (Layer 1) of the SLUM model
- q_2 : The number of QoS attributes at the Service Provider Utility Maximization Layer (Layer 2) of the SLUM model
- q_t : The total number of webservice quality attributes such that $q_t = q_1 + q_2$.
- k : The number of business workflow tasks
- n : The number of functionally similar webservices per workflow task.
- Ω : Theoretical superlinear speedup of SLUM w.r.t S-MIP

B. *Theoretical Superlinear Speedup of SLUM*

Conceptually, the initial set of possible composite services to be optimized using a global planning strategy such as S-MIP is n^k. The time taken to solve the optimization problem in this case is proportional to n^k. Consider that according to [9] and so [24], a webservice is modelled as a vector whose elements are the values of the various QoS attributes. In S-MIP the size of each vector is q_t . Thus, we have *n* candidate QoS vectors per task to be optimized. In SLUM, at layer 1, the size of each vector is q_1 in length, and at layer 2, the size of each (sub) vector is q_2. The size of each (sub) vector at layer 1 as a

proportion of the original vector having q_t elements is $[q_1/(q_1+q_2)]$. Similarly, the size of each (sub) vector at layer 2 as a proportion of the original vector is $[q_2/(q_1+q_2)]$. Respectively, the number of (complete) vectors at layer 1 and layer 2 are $[q_1/(q_1+q_2)\,]*n$ and $[q_2/(q_1+q_2)\,]*n$. Assuming, no service is eliminated at layer 1, the time taken by SLUM to solve the two sequentially decomposed subproblems is proportional to $\left[[[q_1/(q_1+q_2)\,]*n]]^k + [[q_2/(q_1+q_2)\,]*n]]^k\right]$. Thus the relative speedup of SLUM Ω is captured according to (1).

$$\Omega = \left\{ \frac{n^k}{\left[\left[\left[\frac{q_1}{q_1+q_2}\right]*n\right]\right]^k + \left[\left[\frac{q_2}{q_1+q_2}\right]*n\right]\right]^k} \right\} \quad (1)$$

Since $q_t = q_1 + q_2$, (1) is reduced to (2) below.

$$\Omega = [n^k]\,/n^k\{[q_1/q_t]^k + [q_2/q_t]^k\} \quad (2)$$

Equation (2) is further simplified to (3)

$$\Omega = [(c)^k]\,/\{[q_1]^k + [q_2]^k\} \quad (3)$$

If $q_1 \approx q_2$, from (3), then $(q_t)^k > \{[q_1]^k + [q_2]^k\}$ hence $\Omega > 1$. Thus, the complexity of the sum of the parts of two the sequentially decomposed SLUM subproblems grows much slower than the complexity of the whole. Thus, even without elimination of any webservice at layer 1, SLUM would theoretically perform faster than S-MIP. Notice that from equation (3), the speedup due to superlinearity is theoretically independent of the solution space (number of candidate webservices) but dependent upon the total number of QoS attributes (numerator) and upon the number of QoS attributes at the layer 1 and layer 2, and lastly dependent on the size of the business workflow in the number of tasks, *k* (the exponent).

The ideal case is when the ratio $q_1 : q_2 = 1$ **i.e** $q_1 = q_2$, so that the original webservice composition problem is sequentially decomposed into two equal layers in the number of QoS attributes. In this case, equation (3) is further simplified to (4) and finally (5).

$$\Omega = [(q_t)^k]\,/\{[q_1]^k + [q_1]^k\} \quad (4)$$

Given that $q_t = q_1 + q_2$, when $q_1 = q_2 \rightarrow q_t = 2q_1$, so that equation (5) and (6) follow.

$$\Omega = [(2)^k.[q_1]^k\,]\,/\{[q_1]^k + [q_1]^k\} \quad (5)$$

$$\Omega = [(2)^k.[q_1]^k\,]\,/\{[q_1]^k + [q_1]^k\} \quad (6)$$

Hence from (6), we obtain (7)

$$\Omega = (2)^{k-1} \quad (7)$$

(

Thus from (7) we see that provided $q_1 = q_2$, the theoretical superlinear speedup is only dependent on the length of the business workflow and the speedup is a power function of *k.*

Where $q_1 \neq q_2$, we show through the examples below that $\Omega < (2)^{k-1}$ and Ω gets much smaller than $(2)^{k-1}$,tending towards 1 as the ratio $q_1 : q_2$ *or* $q_2 : q_1$ gets much larger than 1. Note that the practical maximum limit of q_1/q_2 is $q_t - 1$, in which case we have $q_t - 1$ QoS attributes at layer 1 and 1 QoS attributes at layer 2. Thus, more formally, let $r = q_1/q_2$ **or** $= q_2/q_1$, whichever is larger . We show that as $r \rightarrow 1, \Omega \rightarrow (2)^{k-1}$ and $r \rightarrow (q_t - 1), \Omega \rightarrow 1$.

Example 1: k =2, $q_1 = q_2 = 4$

This example considers a business workflow with two sequential tasks in which the number of QoS attributes at layer 1 and layer is equal to 4 and therefore q_t =8. Applying the generalized equation in (4) we have $\Omega = [(8)^2]\,/\{[4]^2 + [4]^2\}$=**2.** Since $q_1 = q_2$, we could also use (7) directly to have $(2)^{2-1} = 2$

Example 2: k =2, $q_1 = 4, q_2 = 3$

This example is motivated by the practical considerations of the SLUM model in [24], where the number of QoS at the layer 1 (the Service Consumer Utility Maximization layer) is 4 i.e reputation, security, service execution duration and service access cost, and three QoS attributes at the layer 2 (Service Provider Utility Maximization layer) i.e reliability, availability and throughput. Note that according to the SLUM model [24], the number of QoS attributes at either layer could be varied based on the practical guidelines in [24]. Since $q_1 > q_2, r = \frac{q_1}{q_2} = 1.333$ and $\Omega = [(7)^2]\,/\{[4]^2 + [3]^2\}$=**49/25 = 1.96.** Notice that $r = 1.333 \approx 1$ *and* $\Omega = 1.96 \approx 2$.

Example 3: k =2, $q_1 = 6, q_2 = 1$

This is a hypothetical example where layer 1 has 6 QoS attributes and layer 2 has only one QoS attribute – it demonstrates the effect of a high degree of imbalance between the number of QoS attributes in the two layers on the magnitude of theoretical speedup of SLUM. Here we have $r = q_t - 1 = 6$ and $\Omega = [(7)^2]\,/\{[6]^2 + [1]^2\}$ =**49/43 = 1.13.** Notice that three QoS attributes at the layer 2 (Service Provider Utility Maximization layer) i.e reliability, availability and throughput. Note that according to the SLUM model [24], the number of QoS attributes at either layer could be varied

based on the practical guidelines in [24].Since $q_1 > q_2$, $r = \frac{q_1}{q_2} = 1.333$ and $\Omega = [(7)^2] / \{[4]^2 + [3]^2\}$ =**49/25** = **1.96.** Notice r has reached the limiting value and $\Omega = 1.333$ is much closer to 1 than to $(2)^{k-1}$ or 2.

Notice r has reached the limiting value and $\Omega = 1.333$ is much closer to 1 than to $(2)^{k-1}$ or 2.

IV. EXPERIMENTAL METHODOLOGY

A. Implementation and Experiment Setup

The SLUM and S-MIP algorithms were all implemented in Java 7.0 using the Java Optimization Modeler (JOM)[2] tool version 1.15 with a GLPK[3] linear programming optimization solver. Each of these algorithms invoked a program function called *getBestCompositeService (int optMode)* where "optMode" denotes optimization type.

Benchmark webservice optimization problem instances of varying empirical hardness were randomly generated. Random generation of problem instances is more flexible in adjusting experimental factors and less expensive compared to using real webservice instances. The total number of QoS constraints were fixed at 7. When using SLUM, the number QoS attributes at the Service Consumer Utility Maximization Layer (Layer1) was fixed at 4 as described in [24] and at fixed at 3 Service Provider Utility Maximization Layer (Layer) as described in [24].

Experiments were conducted on a Lenovo computer having Windows Professional 64 bit (6.1, build 7601), Intel Pentium CPU B960, 2 CPUs @2.20 GHz, 2GB RAM, CPU Utilization, Physical Memory Utilization at 1.4 GB of 2GB at any one time.

The number of workflow tasks was fixed at *k=2*. The number of problem instances were 24 and their size in the number of webservices per task varied from 5 to 120 in steps of 5. Before the start of experimentation on any of the problem instances, the QoS constraints at layer 1 were tuned through trial and error such that in each of the problem instances, all candidate webservices at the beginning of layer 1 optimization were promoted to the layer 2 – no elimination of webservices during phase one optimization.

To minimize effects of variance by chance in response time observed on different values of *n*, within and between the two treatments, for every problem instance, *4* consecutive measurements of time were taken one at time and the arithmetic average value recorded. This was achieved by executing the workflow 4 times repeatedly. 4 was chosen because it's statistically known that 4-10 repeated measurements are sufficient to significantly reduce the random errors observed on measurements due to uncertainties in the measurement environment. Moreover, for every problem instance, the time measurement on each of the algorithms was immediately successive. For example, at *n=10*, 4 successive measurements of time are taken on SLUM-MIP, then 10 successive measurements on SLUM, then 4 successive measurements on S-MIP. The process is then repeated for *n=15, n=20* etc. The method differs from the approach where for n=10, 15, 20 .., 4 successive measurements are taken for SLUM for each *n*, then the procedure repeated for S-MIP. The first approach minimizes the time gap between when measurements on the same subject (problem instance) but on a different treatment [25]. This is meant to minimize the impact of intervening system state changes with the passage of time

B. Data Analysis Methodology

Our empirical analysis generally followed the methodology in [25]. Since the authors in [25] statistically established that the CPU runtime growth both SLUM and S-MIP is significantly both polynomial and exponential in nature, here the first step was to fit the data obtained at *k=2* on a quadratic function of the form $\beta_1 n^2 + \beta_0 n + c$ and on an exponential function of the form $\beta_0 e^{\beta_1 n}$. Let the quadratic model for SLUM and S-MIP be $\beta'_{21} n^2 + \beta'_{20} n + c'$ and $\beta''_1 n^2 + \beta''_0 n + c''$ respectively. Let also the exponential regression model of SLUM and S-MIP be respectively $\beta'_0 e^{\beta'_1 n}$ and $\beta''_0 e^{\beta''_1 n}$ accordingly. In the second step, using L-Hospital's rule, under polynomial growth we determine the expected empirical (maximum) speedup of SLUM, SES [25] w.r.t S-MIP as per equation (8). Under exponential growth, using L-Hospital's Law, SES is computed according to equation (9).

$$SES = \beta''_1 / \beta'_1 \quad (8)$$

$$SES = \beta''_0 e^{\beta''_1 n} / \beta'_0 e^{\beta'_1 n} = (\beta''_0 / \beta'_0). e^{(\beta''_1 - \beta'_1) n} \quad (9)$$

The solution to (8) is a constant value while the solution to (9) is an exponential function in *n*. The SES value computed in (8) is compared with the theoretical value 1.96 obtained through the formal analysis in section 3.1.2. It's expected that the empirical value (SES) approximates the theoretical value of 1.96 at *k=2*. Using equation (9), we answer the second research question which is restated as "what is the speedup at *n?"* The value of *n* is substituted in (9). However, the resultant speedup can only be said to be superlinear if SES>1.Thus, it necessary to answer the third research question rephrased as, *what is the minimum value of n for which the value of SES >1?*

[2] *www.net2plan.com/jom/*
[3] https://www.gnu.org/software/glpk/

This is equivalent to solving the inequality $\beta''_0 / \beta'_0) \cdot e^{(\beta''_1 - \beta'_1)n} > 1$. By solving the inequality $(\beta''_0 / \beta'_0). e^{(\beta''_1 - \beta'_1)n} > 1$, we obtain the least value of *n* beyond which a virtual enterprise operating their composite service selection using SLUM start enjoying (the least possible) superlinear speedup of SLUM. We will call the point (value of *n*) beyond which SLUM is expected to start exhibiting superlinear speedup as *SLUM Superlinear Speedup Critical Point* (SSSCP) and denote it by n_{SSC} to contrast it from the n_{CE} point described in [25] that is a generalized expected value of *n* beyond which SLUM's speedup is greater than S-MIP, whether or not service elimination is used. In general, by solving $(\beta''_0 / \beta'_0). e^{(\beta''_1 - \beta'_1)n} >= C$, a virtual enterprise broker could determine the minimum number of service providers (webservices) required to attain at least SES value of *C*. Hence, to compute the minimum value of *n* required to achieve the maximum possible superlinear speedup Ω, we solve (10).

$$\Omega = \beta'_0 \; e^{(\beta''_1 - \beta'_1)n} = \beta''_1 / \beta'_1 \qquad (10)$$

To tackle the fourth research question recast as, *how long would it take to achieve a superlinear speedup SES* at some *n* ? We first need to find the SES as a function of CPU runtime, *t*. Let t_{eB} be the time taken by the SLUM algorithm to solve a problem instance with *n* webservices per task, and t_{eA} be the time taken by SLUM to solve the same problem instance. Since the runtime of SLUM and S-MIP is exponential in nature [25], we use the method of Coffin et al in [23] to plot a scatter plot of *log* t_{eB} *vs log* t_{eA} yields a straight line of the form in equation (10). Removing logs in (11) gives (12). β_0 is the number of times algorithm A is faster than algorithm B initially while $\beta 1$ is the empirical relative complexity coefficient [23]. Asymptotically, $t_{eB} = (t_{eA})^{\beta 1}$ [23].

$$\log t_{eB} = ln\beta'_0 \; + \beta_1 \ln (t_{eA}) \qquad (11)$$

$$t_{eB} = \beta_0 \; (t_{eA})^{\beta 1} \qquad (12)$$

Applying Little Hospital's Rule we have $SES = t_{eA} / \beta_0 \; (t_{eA})^{\beta 1}$ .This leads to equation (12).

$$(\beta''_0 / \beta'_0). e^{(\beta''_1 - \beta'_1)n} = 1/\beta_0 \; (t_{eA})^{1-\beta 1} \qquad (13)$$

We plotted the graph $SES(t_{eA}) = (t_{eA})^{1-\beta 1}$ vs t_{eA} to depict the change in superlinear speedup of SLUM given that S-MIP took t_{eA} to solve some problem instance. We herein call this curve as SLUM Expected Speedup Time Graph (*SESTiG*). It's expected that the growth of the *SESTiG* curve plateaus at the maximum speedup obtained using the formal analysis.

Lastly, using SLUM Sample instantaneous speedup, S_{si} [25], we plotted the graph of S_{si} vs n to show the change in speedup on sample problem instances. We herein refer to the curve as SLUM Sample Instantaneous Scalability Graph *(SSISG)*

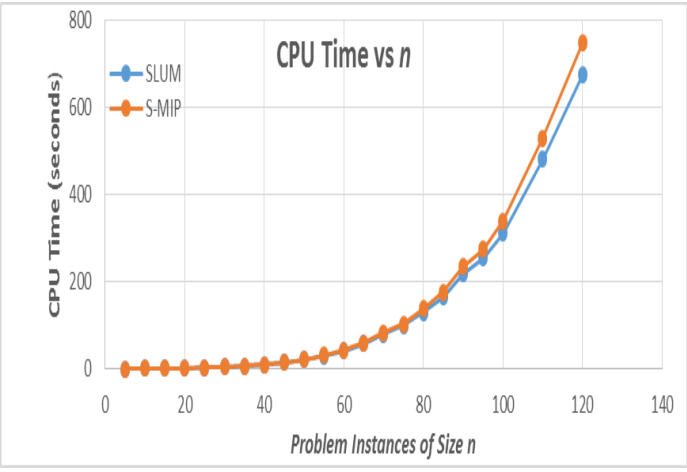
V. RESULTS AND DISCUSSIONS

A. *Observations*

Table 1 presents the CPU runtime performance of SLUM and S-MIP with respect to problem instances of increasing empirical hardness. As explained earlier, optimization inequality constraints were tuned once such that for all problem instances, all candidate webservices evaluated during stage one were all promoted for evaluation in stage 2. The goal was to ensure that any variation in performance between SLUM and S-MIP is not attributed to the service elimination effect. The data shows that until *n=45*, the performance of SLUM is marginally worse than that of S-MIP. Beyond *n=45*, the performance of SLUM is steadily better than that of S-MIP. Moreover the relative speedup of S-MIP *Ssi* increases steadily, starting at 1.017 when *n=45* and grows to 1.2 at *n=120*. The scatter plot in figure 1 and the SLUM Instantaneous speedup curve (SISC) in figure 2 reinforce the observations

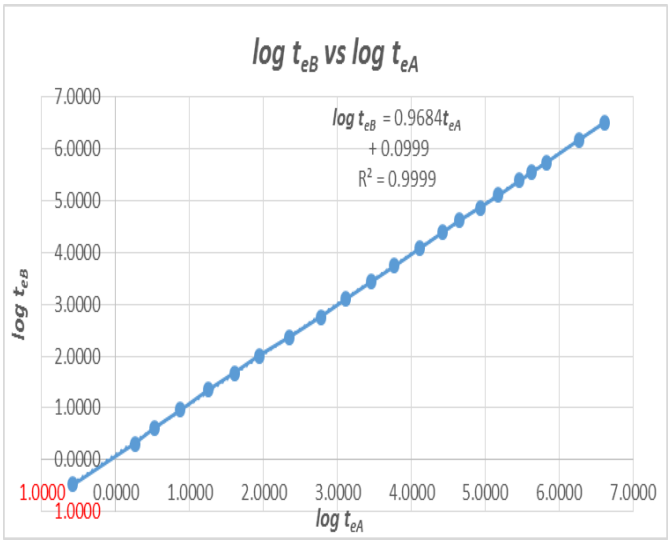
TABLE 1: RUNTIME PERFORMANCE DATA

N	*TB (s)*	*TA (s)*	*Ssi*	*N*	*TB (s)*	*TA (s)*	*Ssi*
5	0.62	0.56	0.9032	60	42.2	43.4	1.0284
10	1.37	1.3	0.9489	65	59.1	61	1.0321
15	1.86	1.7	0.9140	70	79.89	83.1	1.0402
20	2.64	2.4	0.9091	75	100.76	104.3	1.0446
25	3.87	3.49	0.9018	80	130.09	138.2	1.062354
30	5.3	5	0.9434	85	165.54	177	1.069211
35	7.45	6.95	0.9329	90	218.17	235	1.07712
40	10.7	10.5	0.9813	95	254.81	275.6	1.081592
45	15.74	16	1.0165	100	312.01	339.3	1.087455
50	22.2	22.4	1.0090	110	481.76	530	1.100139
55	31	31.6	1.0194	120	673.85	748	1.110038

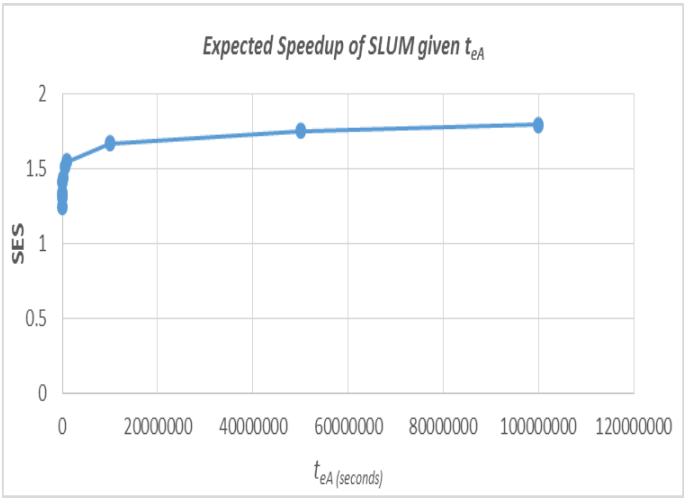


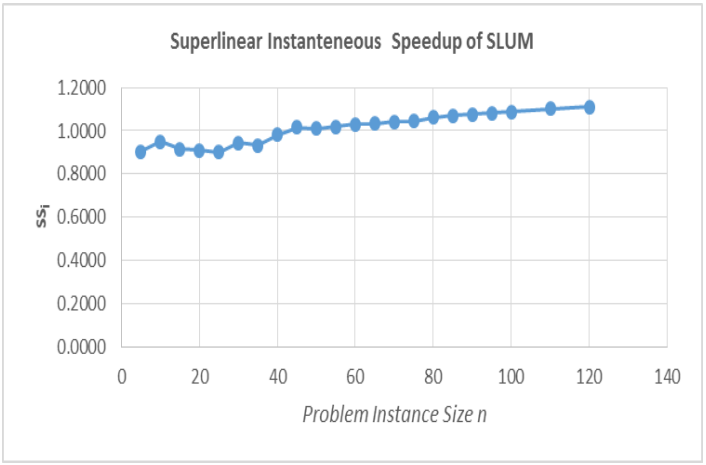
Exponential regression of the curves in figure 1 yielded the equations $t_{eA} = 0.7793e^{0.0624n}$ at $R^2 = 0.9835$ and $t_{eB} = 0.8676e^{0.0605n}$ at $R^2 = 0.9836$. Thus we conclude that both SLUM and S-MIP exhibit exponential growth in running time. By substituting equation (10) with the two exponential equations, and setting determined n_{ssc}=60 as the minimum number of webservices that would be required for SLUM to be at least faster than S-MIP.

The growth behaviour of the two curves in figure 1 hint non constant variance and nonnormality of CPU running time. Thus according to [23], a variance stabilizing transformation (log transformation in this case) is required. The *log-log* scatter plot in Figure 3 being a straight line confirms the heteroskedasticity of the CPU running time. From Figure 3, we infer that the empirical relative complexity coefficient of SLUM w.r.t to S-MIP $\beta_1 = 0.9684$ while the constant term β_0 =1.1 and thus conclude that initially, S-MIP is 1.1 times faster than SLUM but asymptotically, SLUM is more efficient than S-MIP such that $t_{eB} = t_{eA}^{0.9684}$. This means that if for instance the running time of S-MIP is 1000 seconds, the running time of SLUM would be $1000^{0.9684} = 804$ *seconds*.



From figure 3 we obtained SES = $t_{eA}^{.0.0316}$. By plotting $t_{eA}^{.0.0316}$ vs t_{eA} we obtained the graph in figure 4. The graph in figure 4 shows the change in expected relative speedup of SLUM if S-MIP was to take t_{eA} seconds to find an optimal solution to the webservice composition problem. The SES values in figure 4 were generated by evaluating $t_{eA}^{.0.0316}$ for increasingly large values of t_{eA}. The goal was to empirically estimate the limit of the SLUM expected speedup. Figure 4 reveals that the SES values are increasing w.r.t to t_{eA}. However, the same figure reveals that the growth in SES is not infinite, but instead seems to approach a limiting value of 2.





B. *Discussions*

The main research question in this study was "*How does the speedup of SLUM on a set of composite service selection problem instances of increasing hardness change relative to S-MIP in the absence of webservice elimination*?"

From the empirical results, we obtained β_0 =1.1 and β_1 = 0.9684. We conclude that in the absence of webservice elimination, in the initial stages, for small problem instances, SLUM is slower than S-MIP, but in the long run, for large enough problem instances, SLUM is significantly faster than S-MIP. In the initial stages, SLUM incurs the sequential overhead of performing two sequential optimization problems, one at a time, and passing data between the layers [25] compared to S-MIP that executes the optimization problem in one shot, accounting for the slowness. The value β_0 =1.1 obtained in this study is very close to a similar β_0 value of 1.3 obtained in [25]; a very closely related study. On the other hand, even though in this study SLUM does not enjoy the effect of webservice elimination during phase 1 of optimization, the algorithm still records faster performance with growth in problem size complexity, due to the fact that,

when decomposed problems are executed sequentially, the speedup arises from the fact that problem complexity grows more than linearly [13]. This phenomenon is theoretically illustrated in section 3.1.2. Thus our experimental results support our theoretical analysis. Observe that in this study we obtained $\beta_1 = 0.968$ while in our related work in [25], we obtained $\beta_1 = 0.783$. Despite the fact that both β_1 values prove SLUM being relatively faster than S-MIP asymptotically, there is a significant difference between the two values, for example at t_{eA} =1000 seconds at $\beta_1 = 0.968$, SLUM would approximately take 803 seconds to solve the same problem while at t_{eA} =1000 seconds and β_1 =0.783, SLUM would take on average, 224 seconds to solve the same problem instance. This difference could be easily explained using equation (2). In (2), $\Omega = \lceil n^k \rceil / n^k \{ [q_1/q_t]^k + [q_2/q_t]^k \}$ in the absence of webservice elimination. In the presence of service elimination, without loss of generality, assume for every task, ϵ webservices are eliminated during phase 1 of optimization. The number of webservices per task promoted for layer two optimization is $n-\epsilon$ so that in this case

$$\Omega = \frac{n^k}{\left\{ n^k \left[\frac{q_1}{q_t}\right]^k + (n-\epsilon)^k \left[\frac{q_2}{q_t}\right]^k \right\}}$$

Since $(n-\epsilon)^k < n^k$, the value of Ω is generally larger when some services are eliminated compared to when none is eliminated, hence the empirical relative complexity coefficient under service elimination is relatively smaller than that obtained when service elimination used.

From figure 4, it can be seen that the superlinear relative speedup of SLUM w.r.t S-MIP on a two task workflow steadily grows tending towards a maxima value of 2. For instance at t_{eA} *=100,000,000 seconds* is 1.79. This means SLUM can only be 1.79 times faster than S-MIP, if S-MIP would take 1157 days or 3 years to solve some webservice composition problem instance. Since 3 years is too long a time to wait for any practical service request, we therefore conclude that on a two task workflow, the maximum superlinear speedup of SLUM 2. This conclusion is consistent with the formal mathematical analysis in section 3.2 where we showed that under equally decomposed layers, $\Omega = 2^{k-1}$ so that when k=2, $\Omega = 2$. Moreover, from the finding that it would take 3 years for SLUM to be 2 times faster than S-MIP, it implies that in practical sense virtual enterprise brokers won't benefit from the two fold superlinear speedup. However, from figure 4, we also see that at t_{eA}= 100,000 seconds (28 hours), Ω *=1.4.* The significance of this is that a virtual enterprise broker whose service level agreement (SLA) with customers for service delivery is more than 24 hours would be guaranteed to find SLUM 1.4 times more efficient than if they used S-MIP, even in instances where the optimization parameters are such that no service elimination took place.

The minimum size of a workflow in the number of webservices per task required for SLUM to be faster than S-MIP without service elimination has been determined as 60. In contrast, the equivalent workflow size when elimination is present as computed in [Abiud] is 22. The significance of this finding is that at time instances where optimization parameters are such that few or no service is eliminated, only virtual enterprise broker operating more than 61 service providers per workflow task would experience SLUM as being more efficient than S-MIP, while whenever some services are eliminated in phase 1, virtual enterprise brokers having as few as 22 virtual enterprise service providers per workflow task, will find SLUM more efficient than S-MIP in meeting their dynamic webservice composition needs.

VI. CONCLUSION

A. Contributions

Through formal analysis, the key theoretical contributions of the paper are as follows. We have shown that even without service elimination at layer 1, for a business workflow with *k* sequential tasks, SLUM is still faster than S-MIP by 2^{k-1} times on the maximum, provided the number of QoS attributes in layer 1 and in layer 2 are approximately equal. We have also shown that in the case of the number of QoS attributes in the two SLUM layers being unequal then the speedup is given by $\Omega = \lceil (q_t)^k \rceil / \{ [q_1]^k + [q_2]^k \}$. We have further experimentally validated the theoretical analysis and established that on a two workflow the relative speedup of SLUM with respect to S-MIP without service elimination has a limit of 2**.** Moreover, we have empirically demonstrated that even though a theoretical speedup of 2 is possible, it's practically impossible to achieve as it would require more than 3 years to be achieved. Instead our experimental results show that SLUM guarantees a superlinear speedup of about 1.5 times within 28 hours, a time duration that could be practically acceptable to certain types of virtual organizations e.g those dealing in virtual supply chain management.

A methodological and conceptual contribution in this study is the definition of SLUM Sample Scalability Graph (SSISG) and SLUM Expected Speedup Time Graph (*SESTiG*) as complementary methods of performance data analysis. The SSIG builds from our previous work in [] that defines the concept of "SLUM Sample Instantaneous Speedup". The SSIG visually depicts relative speedup of SLUM w.r.t to the growing size of sample optimization problem instances. On the other hand, *SESTiG* is derived from the concept of empirical relative complexity coefficient as defined in [23]. From the empirical relative complexity coefficient, we derived an expression of the expected speedup of SLUM as a function of time of the form $t_{eA}{}^{\mu}$. *SESTiG* is the graph $t_{eA}{}^{\mu}$ *vs* t_{eA} . Using the *SESTiG*, one can predict the growth behaviour of the relative speedup of algorithm *B* w.r.t *A* as a function of the time taken by algorithm to solve some problem instance drawn from the population. A related contribution is that one can use

the *SESTiG* to determine and or verify the maximum speedup limits of an algorithm *B* w.r.t an algorithm *A.* For example, in this study, we first theoretically determined the maximum possible superlinear speedup of SLUM to be 2 for a two task workflow. We then went ahead as illustrated in figure 4 to empirically verify using the *SESTiG* that the theoretical speedup limit practically holds.

From the mathematical analysis and the experimental results, the study also makes a significant practical contribution to the state of the art. As determined in the previous sections, the general expression for SLUM relative speedup is:

$$\Omega = \frac{n^k}{\left\{ \begin{matrix} n^k \left[\frac{q_1}{q_t}\right]^k + \\ (n-\epsilon)^k \left[\frac{q_2}{q_t}\right]^k \end{matrix} \right\}}$$

So that when $\epsilon=0$ then no service elimination took place at layer 1 for a particular problem instance and $\epsilon>1$ means some services were eliminated during phase 1. The exact value of ϵ is determined by a combination of optimization parameters such as the webservice QoS matrix values, the size of the QoS matrices, the constraint inequality expressions and the values on the right hand side of the constraint inequalities. The study in [25] focused on the general case where service elimination at all times is assumed. The study then went ahead to prove that beyond $n=22$, SLUM is consistently faster than S-MIP. However, considering the dynamic nature of the service environment, the likelihood of $\epsilon=0$ is not a rarity. In such cases, the virtual enterprise broker would be interested in determining the worst case relative speedup guarantees of SLUM w.r.t S-MIP and under what conditions. To the best of our knowledge, no previous study addresses this concern. Our contribution is that when $\epsilon=0$ and k=2, only virtual enterprise brokers operating at least 60 providers per task can gain from using SLUM as opposed to S-MIP. In addition, we have proven that the maximum practical superlinear speedup that virtual enterprise brokers can enjoy from SLUM is 1.5 (or 50% gain in computational speed) as opposed to the theoretical maximum value of 2 (or 100% gain in computational speed).

B. Future Work

The equation $\Omega = 2^{k-1}$ suggests that theoretically, the maximum superlinear speedup of SLUM w.r.t S-MIP increases with the size of the business workflow int the number of sequential tasks. Experimentally validating this theory would be worthwhile.

REFERENCES

[1]. Molina A. and Flores M., "A Virtual Enterprise in Mexico: From Concepts to Practice", Journal of Intelligent and Robotics Systems, 26: 289-302, 1999.

[2]. Amit G., Heinz S .and David G. (2010). Formal Models of Virtual Enterprise Architecture: Motivations and Approaches, PACIS 2010 Proceedings.

[3]. Cammarimha M. and Arfsamanesh .(2007). A comprehensive modelling framework for collaborative networked organizations, Journal of Intelligent Manufacturing, Springer Publisher,Vol.18 (5),pp-527-615.

[4]. Arfsamanesh H. et al (2012). A framework for automated service composition in collaborative networks, FNWII: Informatics Institute, http://hdl.handle.net/11245/1.377/099.

[5]. Dustdar S. and Wolfgang S. (2005). A Survey on Web Services Composition.

[6]. Kuyoro Shade O. et al (2012). Quality of Service (QoS) Issues in Web services, International Journal of Computer Science and Network Security, Vol 12 (1), January, 2012.

[7]. Mahboobeh M. and Joseph G.D (2011). Service Selection in Web service Composition. A comparative Review of Existing Approaches, Springer-Verlag Berlin, Heidelberg, 2011.

[8]. Rajendran T. and Balasubramanie P. (2009). Analysis on the Study of QoS Aware Web services Discovery, Journal of Computing Vol. 1(2), December, 2009.

[9]. Benatallah B. et al (2005). QoS-Aware Middleware for Web Services Composition. IEEE Transactions on Software Engineering, Vol. 30, No. 5, May 2005.

[10]. Bin Xu et al (2011). Towards Efficiency of QoS driven semantic web service composition for large scale service oriented systems, Sringer, 211, DOI 10.1007/s11761-011-0085-8.

[11]. Singh K.A (2012). Global Optimization and Integer Programming Networks. International Journal of Information and Communication Technology Research.

[12]. Peter Bartalos, M´ariaBielikova (2011). Automatic Dynamic Web Service Composition: A Survey and Problem Formalization, Computing and Informatics Journal, Vol. 30, 2011, 793–827.

[13] Stephen Byod, Lin Xiao and AlmirMutapcic (2003). Notes on Decomposition Methods, Notes for EE3920, Stanford University, Autum 2003 available at http://web.stanford.edu/class/ee392o/decomposition.pdf, Last accessed 30th December, 2015.

[14]. Chiang Mung. (2006). Layering as Optimization Decomposition, Electrical Engineering Department, Princeton University. Also available online at http://www.ece.rice.edu/ctw2006/talks/ctw06-chiang.pdf.

[15]. Chiang M. et al. (n.d). Layering as Optimization Decomposition. Current Status and Open Issues, Electrical Engineering Department, Princeton University.

[16]. Chiang M. et al. Layering as Optimization Decomposition. Ten Questions and Answers, available athttp://web.stanford.edu/class/ee360/previous/suppRead/read1/layer_1.pdf.

[17]. Steve Low (2013). Scalable Distributed Control of Networks of DER, Computing & Math Sciences and Electrical Engineering, Caltech University.

[18]. Virginie G. et al (2013). A linear Program for QoS web service composition based on complex workflow. 2013.

[19]. Ardagna, D. and B. Pernici, *Adaptive Service Composition in Flexible Processes.* IEEE Trans. on Software Eng., 2007.

[20]. Rabelo R., Gusmeroli S. The ECOLEAD collaborative business infrastructure for networked organizations networked organizations. Pervasive collaborative networks PRO-VE 2008. Springer, New York, 2008.

[21] . Pervasive collaborative networks PRO-VE 2008. Springer,33 (6): p. 369-384.

[22] Susan D. Urban and Le Gao. (xxxx). A Survey of Transactional Issues for Web Service, Composition and Recovery, Int. J. Web and Grid Services, Vol. x, No. x, xxx.

[23] . Marie Coffin and Mathew J. Saltzman (2000). Statistical Analysis of Computational Tests of Algorithms and Heuristics, INFORMS Journal on Computing , Vol. 12, No. 1, Winter 2000.

[24]. Abiud Wakhanu. Mulongo, Elisha T. Omulo and William O. Odongo (2015). A Hierarchical Multilayer Service Composition Model for Global Virtual Organizations, Computer Science and Information Technology 3(4):91-104, 2015.

[25]. Abiud Wakhnau. Mulongo, Elisha T. O. Opiyo, Elisha Abade, William Okello Odongo, Bernard Manderick (2016): SLUM: Service Layered Utility Maximization Model to Guide Dynamic Composite Webservice Selection in Virtual Organizations, Computer Science and Information Technology Vol. 4, No. 2 , 2016. In Press. .

[26]. Rabelo J. Ricardo et al (2007). An Evolving Plug and Play Business Infrastructure for Networked Organizations. International Journal of on Information Technology and Management, 2007.