Towards a Bell-Curve Calculus for e-Science

Lin Yang[1] **Alan Bundy**[1] **Dave Berry**[2] **Conrad Hughes**[2]

[1] University of Edinburgh [2] National e-Science Centre

Abstract

Workflows are playing a crucial role in e-Science systems. In many cases, e-Scientists need to do average case estimates of the performance of workflows. Quality of Service (QoS) properties are used to do the evaluation. We defined the Bell-Curve Calculus (BCC) to describe and calculate the selected QoS properties. The paper presents our motivation of using the BCC and the methodology used during the developing procedure. It also gives the analysis and discussions of the experimental results from the ongoing development.

1. Introduction

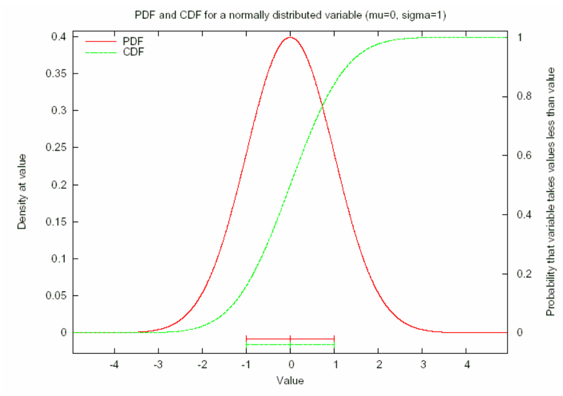
Grid computing has an almost ten-year history since it was derived, from an analogy to the power Grid, to denote a proposed distributed computing infrastructure for advanced science and engineering collaborations [1]. It is strongly required by consumers, scientists, engineers, enterprises, countries, and even the whole world to share resources, services and knowledge [2]. This sharing is supported and implemented by web services, software systems designed to support interoperable machine-to-machine interaction over a network. These services can be composed in many different ways to form workflows. It is very helpful to measure the performance of the resulting composite services because their quality affects the quality of the Grid directly.

In scientific workflows, experimental data is generated and propagated from one service to another. It would be useful to get rough estimates of various QoS properties, e.g. reliability, accuracy, run time, etc. so that e-Scientists could perform analyses and evaluations of either services or the data produced. We have previously thought about the use of interval arithmetic to calculate error bounds on such estimates. The idea is to extend a numeric value to a number interval, e.g. we use an interval, say [41, 43], to represent the range of error of 42. Extended numeric analysis is used as the way of interval propagation in workflows. The simplest example is for a unary and monotonically increasing function f(x), the extended function f*([a, b]) = [f (a), f (b)]. Using interval arithmetic and propagating error bounds will calculate the biggest accumulative error during workflow executions, so it is a good method for doing a *worst-case* analysis.

However, in more common situations, e-Scientists may want to know the likelihood of each value in the interval. So for *average-case* analysis, we propose to use normal distributions (bell curves) to add the concept of probability to differentiate the likely from the unlikely values of the QoS properties. That is, if we associate a probability density function (pdf) shaped as a bell curve with the estimate, then some values in the interval have a higher probability than others. Figure 1 defines and illustrates the pdf and cumulative density function (cdf) of a standard bell curve.

FIGURE 1

STANDARD BELL CURVE



The graph shows a standard bell curve with parameters – mean value μ=0 and standard deviation σ=1. The red curve is the pdf (probability density function) curve, indicating the probability of each possible value of variable x. It can be generally presented as $p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$. The green curve is the cdf (cumulative density function) curve, integrated from its pdf. It gives the probability that a normally distributed variable will output a value ≤ x.

So now the questions are:

(1) Can we use BCC to describe QoS properties and what are the advantages and disadvantages?

(2) How can we define a BCC?

We aim to prove the hypothesis:

The Bell-Curve Calculus is a good estimate of Quality of Service properties over a wide range of values.

2. Why a Bell-Curve Calculus

Although PEPA [3] and some other projects use exponential distribution as their atomic distribution, we still have sufficient reasons to choose bell curve. Initial experimental evidence from DIGS [1] suggests that bell curves are a possible approximation to the probabilistic behaviour of a number of QoS properties used in e-Science workflows, including the reliability of services, considered as their mean time to failure; the accuracy of numerical calculations in workflows; and the run time of services. Moreover, the Central Limit Theorem (CLT) [4] also gives us some theoretical support by concluding that:

"*The distribution of an average tends to be Normal, even when the distribution from which the average is computed, is decidedly non-Normal.*"

Here in the CLT, 'Normal' refers to a Normal Distribution, i.e. a Bell Curve.

Furthermore, from the mathematical description of bell curves, we can see that the biggest advantage of using a bell curve is that its probability density function (pdf) p(x) $= \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$ has only two parameters: mean value μ and standard deviation σ, where μ decides the location of a bell curve and σ decides the shape of a bell curve. While evaluating the performance of a workflow, we need to gather all the possible data values of the QoS properties we analyse from all the input services. We do calculations and analysis using the information and pass the results through the whole workflow. It will be a big burden if we transfer and calculate all the possible data values one by one. Now using bell curves which have only two parameters, the job becomes more efficient. All we need to do is to store and propagate the two parameters in workflows and a bell curve can be constructed at any time from μ and σ.

Then we will see if we can calculate the QoS properties of a whole workflow from the corresponding properties of its component services, namely if we can define some inference rules to derive the QoS properties of composite services from the correlative properties of their components.

We consider four fundamental methods to combine Grid services (we use services s_1 and s_2 to represent two arbitrary services).

Sequential: s_2 is invoked after s_1's invocation and the input of s_2 is the output of s_1.

Parallel_All: s_1 and s_2 are invoked simultaneously and the outputs are both passed to the next service.

Parallel_First: The output of whichever of s_1 and s_2 first succeeds is passed to the next service.

Conditional: s_1 is invoked first. If it succeeds, its output is the output of the workflow; if it fails, s_2 is invoked and the output of s_2 is the output of the whole workflow.

In terms of the three QoS properties and four combination methods, we have twelve fundamental combination functions (see Table 1). For instance, the combination function of run time in sequential services is the sum of the run times of the component services.

TABLE 1
THE TWELVE FUNDAMENTAL COMBINATION FUNCTIONS

	Seq	**Para_All**	**Para_Fir**	**Cond**
run time	sum	max	min	cond1
accuracy	mult	combine1	varies?	cond2
reliability	mult	combine2	varies?	cond3

The table shows the twelve fundamental combination functions in terms of three QoS properties and the four basic combination methods. Sum, max, min and mult represent respectively taking the sum, maximum, minimum and multiplication of the input bell-curves. Cond1-3 are three different conditional functions and their calculation depends on the succeeding results. The functions of Varies are parallel_first, which means the output of the workflow is the output of the first succeeded service. Combine1-2 are probabilistic merges, which are in the forms of linear-_combinations_of_distribution_1*probability_of_1_occurring+linear_combinations_of_distribution_2*probability_of_2_occurring+...+linear_combinations_of_distribution_N*probability_of_N_occurring. Neither are uniquely defined functions, but depend on different use cases of workflows, which adds the uncertainty to the calculus. But in most workflows, only the basic combinations sum, max, min and mult are needed. What we do for combine1-2 is to combine these basic functions based on different workflow.

Here we convert the formula of bell curve to a function in terms of μ and σ, then get the bell curve function as

[1] DIGS (Dependability Infrastructure for Grid Services, http://digs.sourceforge.net/) is an EPSRC-funded project, to investigate in fault-tolerance system and other quality of service issues in service-oriented architectures

$$bc(\mu,\sigma) = \lambda x. \frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\sqrt{2\pi}\sigma}$$

.

Our job is to define different instantiations of the combination functions applying to different QoS properties and different workflow structures.

3. Methodology

Suppose we have two bell curves corresponding to two services. We present them using a bell curve function defined in Section2 as $bc(\mu_1, \sigma_1)$ and $bc(\mu_2, \sigma_2)$. We need to describe μ_0 and σ_0 using μ_1 , μ_2 , σ_1 and σ_2 . That is, μ_0 = $f_\mu(\mu_1, \mu_2, \sigma_1, \sigma_2)$ and σ_0 = $f_\sigma(\mu_1, \mu_2, \sigma_1, \sigma_2)$ The combination function $bc(\mu_0, \sigma_0)$ is defined as $bc(\mu_0, \sigma_0)$ = $F(bc(\mu_1, \sigma_1), bc(\mu_2, \sigma_2))$ = bc($f_\mu(\mu_1, \mu_2, \sigma_1, \sigma_2)$, $f_\sigma(\mu_1, \mu_2, \sigma_1, \sigma_2)$), which is actually a function in terms of four parameters -- μ_1, μ_2, σ_1 and σ_2.

Therefore we have two main tasks:

(1) Can we find a satisfactory instantiation of $F(bc(\mu_1, \sigma_1), bc(\mu_2, \sigma_2))$ for every situation we are investigating?

(2) How good will our approximations be? 'Good' here means accurate and efficient.

For example, for the property run time in sequential services, we can use $\mu_0 = \mu_1 + \mu_2$ and $\sigma_0 = \sqrt{\sigma_1^2 + \sigma_2^2}$, which has been proved true in mathematics [5].

Our experiments are based on a system called Agrajag [2]. Using Agrajag, we got a satisfactory match (the error is generated by the limited calculation in the approximation method in Agrajag) of the piecewise uniform approximation curve (blue curve) and our estimate curve (mauve curve) (see Figure 2).

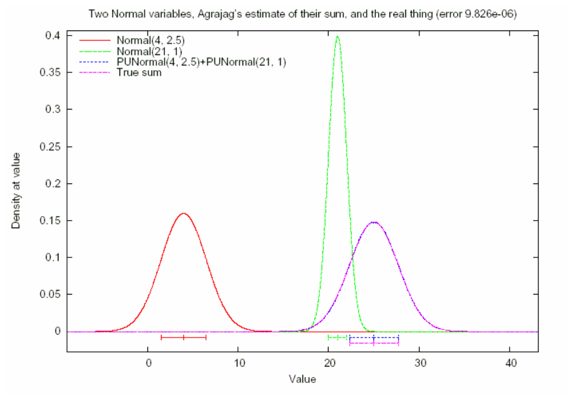
Some of our combination functions have been defined by ourselves and tested in Agrajag. For example, for runtime in parallel_all structure, we need to get the maximum of two bell curves. Figure 3 shows the situation of the maximum of two bell curves using the combination method: $\mu_0 = \max(\mu_1, \mu_2)$ and $\sigma_0 = \max(\sigma_1, \sigma_2)$. In this graph, we can see that our estimate achieved a good result – the error is very small. But does it always work like this? When we choose two closer bell curves as the inputs, the error became comparatively large (see Figure 4). This inconsistency decided one aspect worth investigation: through systematic experimentation using Agrajag, we needed to explore in a wide range of data to find various error status in different input situations.

[2] Agrajag (http://digs.sourceforge.net/agrajag.htm) is a framework written in Perl and C, developed by Conrad Hughes, to implement some operations and measurements on some basic models of stochastic distributions.

FIGURE 2

THE SUM OF TWO BELL CURVES

AND ITS APPROXIMATION

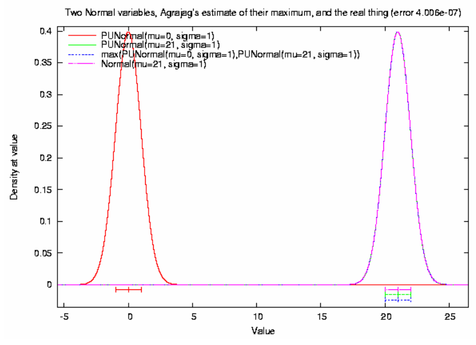


The graph shows the sum of two bell curves (red curve and green curve). It can be used to model the run time of sequential combinations. Here we use an exact mathematically proved method: $\mu_0 = \mu_1 + \mu_2$ and $\sigma_0 = \sqrt{\sigma_1^2 + \sigma_2^2}$ to estimate the piecewise uniform curve (blue curve) produced by Agrajag. The mauve curve is our approximation curve, which almost coincides with the blue curve. We can see there is still a tiny error shown at the title of the graph. It is caused by the approximation using piecewise uniform functions. In the ideal situation (the resolution values which divide a curve to locally constant and connected segments $\rightarrow +\infty$), the error is zero.

FIGURE 3

THE MAXIMUM OF TWO BELL CURVES

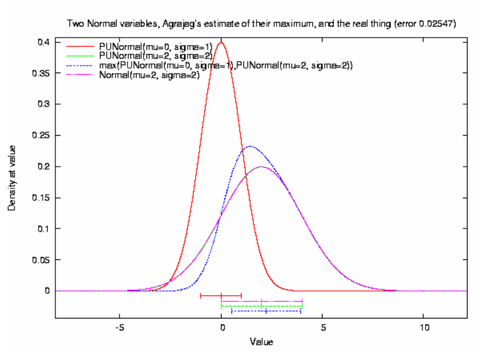
AND ITS APPROXIMATION (1)



This graph shows an ideal situation of getting the maximum of two bell curves. The red curve and the green curve are the two inputs. In this case, using the method $\mu_0 = \max(\mu_1, \mu_2)$ and $\sigma_0 = \max(\sigma_1, \sigma_2)$, the green curve is the piecewise uniform form of our approximation, the mauve curve. The blue curve is Agrajag estimate. Since the green curve, the blue curve and the mauve curve almost coincide with each other, they are hardly distinguished in the figure.

FIGURE 4

THE MAXIMUM OF TWO BELL CURVES AND ITS APPROXIMATION (2)



In this graph, we use the same combination method as that in Figure 3, but taking two much closer bell curves as the inputs. This time, there is a distinct difference between Agrajag's estimate (the blue curve) and our approximation (the mauve curve). We can see that in this case, the mauve curve and the green curve are the same curve, so they coincide with each other.

Another investigation aspect is to define and compare all sorts of combination methods to get, say, the maximum of bell curves. For example, through testing in Agrajag, we discovered that in most common situations, to get the maximum of two bell curves, the effect of approximating the output curve using $\sigma_0 = \max(\sigma_1, \sigma_2)$ is better than that using $\sigma_0 = \sqrt{\sigma_1^2 + \sigma_2^2}$ or that using $\sigma_0 = 1/(1/\sigma_1 + 1/\sigma_2)$. Our main goals are to make comparisons of all sorts of approximation methods and find the best one across many different situations.

To achieve a better outcome, we need some methods to define the precision of the approximation methods we use and then refine the experimental results. The explicit way to get to precision is to calculate the average error values, which allows us to have a general idea about how accurate our approximation is and make a comparison between different approximation methods easily. However, it cannot indicate how we could improve our method to get a better result. In Agrajag, there is a functionality to derive the parameters of the piecewise uniform approximation to the combination functions. So we call these parameters the perfect parameters and use them as a standard. Then we transform the job of finding the most suitable parameters of the combination functions to matching the perfect parameters. We will elaborate it using an example in Section 4.

All the above description to our methodology raises the question: since Agrajag can perform piecewise uniform approximation of bell-curve combinations, why do we still need a BCC? Why don't we just use Agrajag to produce a bell-curve approximation to a workflow using the data from its component services? The answer is efficiency. Agrajag's calculations do well in small workflow calculations, but the more common scenario is that workflows sometimes are composed of thousands of services. To take all the inputs and get an approximation requires huge calculation capacity, which will make Agrajag's runtime unacceptably long. While using the BCC, we just need to do calculations among the parameters, which will make the calculation procedure more efficient.

4. Experimental Result and Analysis

In this section, we will give some experimental results and analysis according to the methodology we have described in Section 2 and Section 3. Since the combination function of sum of two bell curves is exact, we make our first attempt on the method of the getting maximum of two bell curves, which does not have a known simple mathematical combination.

To get more intuitive results, we used Gnuplot[3] to draw 3D graphs. Without loss of generality, we fixed one of the input bell curves to the standard bell curve (μ_2=0 and σ_2=1). Then the three dimensions were set as μ_1, σ_1 and the difference between the piecewise uniform estimation and our approximation using our combination methods. To ensure that common situations are considered, we generated $bc(\mu_1, \sigma_1)$ from a range of logarithmic-scaled integers, e.g., $2^{-10} \le \mu_1 \le 2^{10}$ and $2^{-10} \le \sigma_1 \le 2^{10}$.

Figure 5 shows the experimental results using a combination method (Method 1): $\mu_0 = \max(\mu_1, \mu_2)$ and $\sigma_0 = \max(\sigma_1, \sigma_2)$. From this graph we can see how the value of the error changes. Especially in the area of $7 \le \mu_1 \le 9$ and $1 \le \sigma_1 \le 1.6$, the errors are near 1e-06, which is a quite satisfactory approximation.

Does the method shown in Figure 5 achieve the best result? We tested another method (Method 2): $\mu_0 = \max(\mu_1, \mu_2)$ and $\sigma_0 = \sigma_1$ or σ_2 (with bigger μ) (see Figure 6). In Figure 6, we can see that the area of tiny errors is extended, compared to Figure 5. In most areas, the two surfaces coincide with each other, which is always true when $\sigma_1 \ge 1$ because $\sigma_2 \equiv 1$ and both

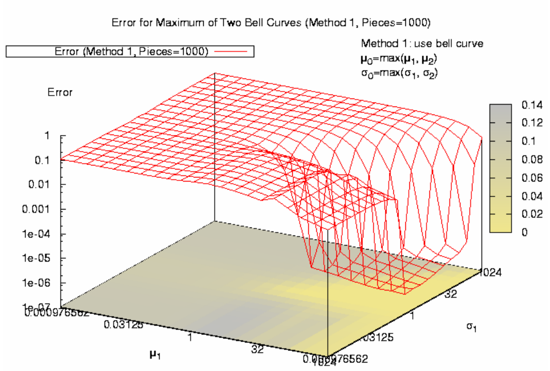
[3] Gnuplot is a portable command-line driven interactive data and function plotting utility for many operating systems. It can plot either 2D or 3D graphs.

methods will take $\sigma_0 = \sigma_1$. Whereas in the area $\mu_1 \geq 4$ and $\sigma_1 < 1$, the green surface (Method 2) is much lower than the red one (Method 1). But the two methods are still the best two among all the methods we tried. Table 2 shows all the combination methods we had tried to get the maximum of two bell curves and their average errors. For all the methods we used $\mu_0 = \max(\mu_1, \mu_2)$.

FIGURE 5

THE MAX OF TWO BELL CURVES

AND ITS APPROXIMATION – METHOD 1

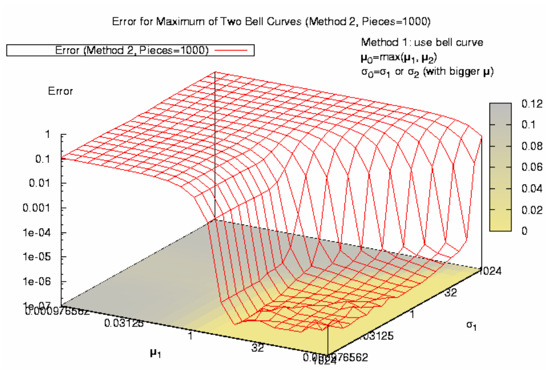


The graph shows the error distribution for $2^{-10} \leq \mu_1 \leq 2^{10}$ and $2^{-10} \leq \sigma_1 \leq 2^{10}$. The X-axis is the value of μ_1, the Y-axis is the value of σ_1 and the Z-axis is the value of error between the piecewise uniform estimate (the resolution values is 1000 for this case) and the BCC estimate using the method $\mu_0 = \max(\mu_1, \mu_2)$ and $\sigma_0 = \max(\sigma_1, \sigma_2)$. A lower value for the Z-axis shows a better fit. The values of x, y and z are discrete, but we set it drawn with the parameter 'by steps', which allows Gnuplot to connect every two points and give us a clearer figure. The colourful platform map at the bottom of the coordinates indicates the various values of error. From the label on the right, we can see that from grey to yellow, the value of error decreases.

FIGURE 6

THE MAX OF TWO BELL CURVES

AND ITS APPROXIMATION – METHOD 2



The graph shows the error distribution using the method $\mu_0 = \max(\mu_1, \mu_2)$ and $\sigma_0 = \sigma_1$ or σ_2 (with bigger μ). Please note that the yellow areas in the platform map do not imply that all the values of the error are zero, but rather the errors are too small to distinguish from zero.

When we observe the above three figures, we can see that the errors produced by both methods stay stable at a comparatively high value in some areas. For instance, in Figure 6, there are some areas with correspondingly high error values and a sharp descent on error values at $\mu_1 \approx 4$. Why is there a distinct difference among the values? We did an experiment using method 2 to get the answer.

We set the numbers of pieces of piecewise uniform functions as 10, 100 and 1000 and got the three piecewise uniform estimates of the maximum of two bell curves. Then we used method 2 to derive our approximation of the maximum and obtained three error distributions. We drew the three distributions in one graph (Figure 7). We can see that the high-error areas of the three distributions coincided with each

TABLE 2

THE COMBINATION METHODS OF GETTING MAXIMUM OF TWO BELL CURVES

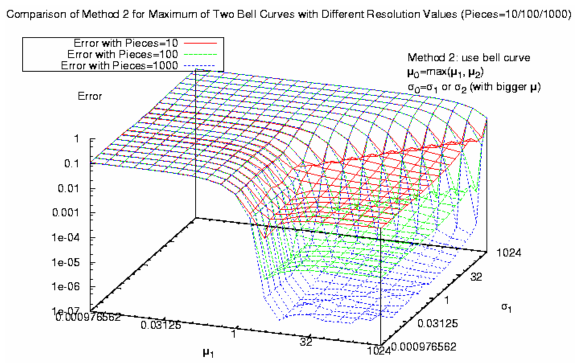
Method	Average error	Method	Average error
$\sigma_0 = \max(\sigma_1, \sigma_2)$	0.0563955	$\sigma_0 = \sigma_1$ or σ_2 (with bigger μ)	0.0550807
$\sigma_0 = \sqrt{\sigma_1^2 + \sigma_2^2}$	0.0782595	$\sigma_0 = 1/(1/\sigma_1 + 1/\sigma_2)$	0.0799015
$\sigma_0 = \sigma_1 + \sigma_2$	0.0862074	$\sigma_0 = 1/\sigma_1 + 1/\sigma_2$	0.1160483
$\sigma_0 = 0.8 \times \sigma_1 + 0.2 \times \sigma_2$ (with bigger μ)	0.0564251	$\sigma_0 = \sqrt{(\sigma_1^2 + \sigma_2^2)/2}$	0.0676253
$\sigma_0 = \sigma_1 \times \sigma_2$	0.0550807	$\sigma_0 = \sqrt{\sigma_1 \times \sigma_2}$	0.0648305
$\sigma_0 = (\sigma_1 + \sigma_2)/2$	0.0650388	$\sigma_0 = \lvert\sigma_1 - \sigma_2\rvert \times \sigma_1/\sigma_2 + \lvert\sigma_1 - \sigma_2\rvert \times \sigma_2/\sigma_1$	0.1194134

The table shows the situation when we use different combination methods to get the maximum of two bell curves at the same resolution value. We set $\mu_0 = \max(\mu_1, \mu_2)$ in all the methods. μ_1 And σ_1 both take values from 2^{-5} to 2^{5}. Since we use the standard bell curve as one input and $\mu_1 > 0$, $\sigma_0 = \sigma_1$ or σ_2 (with bigger μ) and $\sigma_0 = \sigma_1 \times \sigma_2$ are the same method in this case. Despite this, the first two combination methods are the best two methods we got. The combination methods we choose are rough hypotheses based on Figure 4. We estimate the output parameters according to the location and shape of the Agrajag approximation curve. We calculated the average error of each method to compare how good these combination methods are.

other. But since the three distributions used different resolution values, which means that the precisions of the three calculations are different and there should be a minimum difference on the error values with different numbers of pieces. While in some areas, the value is almost unchanged, which means the method we used did not get a correct result in these areas. We tested all the methods we used in our experiment and could not find a completely satisfactory method.

FIGURE 7

THE COMPARISON OF COMBINATION METHOD 2 WITH DIFFERENT RESOLUTION VALUES



The graph provides us the comparison of error distributions when we separately take the number of a pieces in piecewise uniform estimate as 10, 100 and 1000. Here the method is $\mu_0 = \max(\mu_1, \mu_2)$ and $\sigma_0 = \sigma_1$ or σ_2 (with bigger μ).

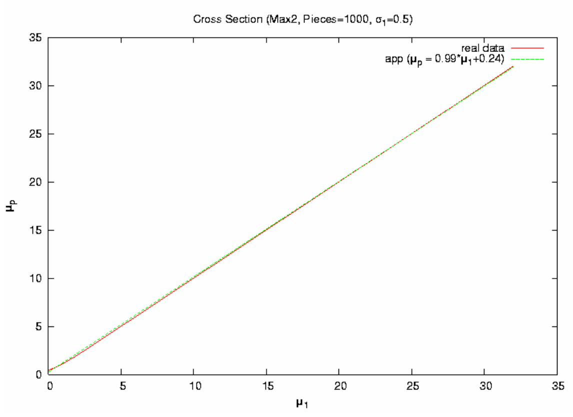
In this case, we could use our perfect parameters method stated in Section 3 to facilitate fine adjustments on our approximation functions.

We derive the perfect parameters μ_p and σ_p in Agrajag and try to approximate them in terms of μ_1, μ_2, σ_1 and σ_2. Then the combination function we aim for turns to $bc(\mu_p, \sigma_p)$ = bc($f_\mu(\mu_1, \mu_2, \sigma_1, \sigma_2)$, $f_\sigma(\mu_1, \mu_2, \sigma_1, \sigma_2)$) $F(bc(\mu_1, \sigma_1), bc(\mu_2, \sigma_2))$.

We fix one parameter, such as σ_1, then use some function to describe the relation between μ_p and μ_1. To simplify the problem, we use linear function here, namely $\mu_p = a \times \mu_1 + b$. Then we get sets of a and b corresponding to different σ_1 value. Figure 8 gave us the linear approximation when σ_1 = 0.5. Then we could have the linear function of σ_1 in terms of a and b. So we finally get the function of μ_p in terms of μ_1 and σ_1 (see Figure 9).

FIGURE 8

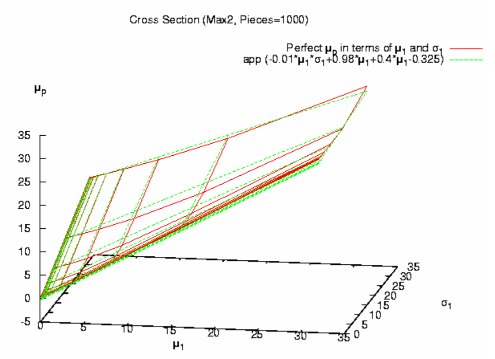
THE LINEAR APPROXIMATION OF μ_p



The red line is the real data of μ_p and the green line is our approximation using the linear function $\mu_p = 0.99 \times \mu_1 + 0.24$. In this case, σ_1 is fixed to 0.5. In this figure, the linear approximation achieved a good result, especially in the upper part of the curve.

FIGURE 9

THE APPROXIMATION OF μ_p IN TERMS OF μ_1 AND σ_1

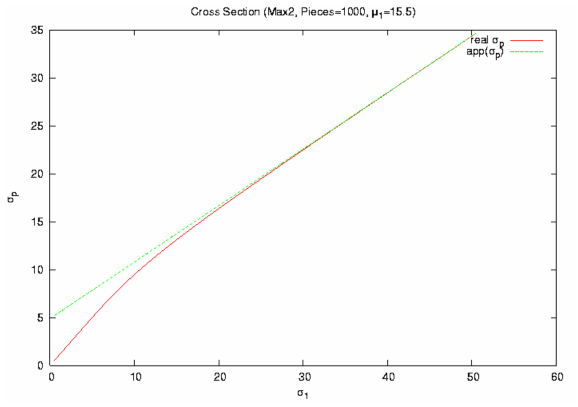


In this graph, the red surface is the real perfect μ_p surface and the green surface is our approximation of μ_p in terms of μ_1 and σ_1. We can see in this figure, the two surfaces do not coincide in all the areas, which means that we still need to do some adjustment during the procedure of the linear approximation. We could change parameters of the linear functions or use non-linear functions to do the approximation.

In the above example, only using linear approximation seems to get a good result. However, in most cases, linear functions alone cannot achieve a satisfactory result. For example, this time we fix μ_1 to 15.5, then we use a linear function of σ_1 to approximate σ_p (see Figure 10). There is a big gap between the curve and the approximation. After drawing the difference curve, based on the curve shape, we used different compensation to approximate it. The choices of exponential or other functions were made by our experience and the parameters of those functions were chosen by fitting the curves gradually. Figure 11 shows the result after adding the exponential compensation. The error between the real perfect values and our approximation is reduced appreciably. We could repeat the above procedure until the result reaches the acceptable range.

FIGURE 10

THE LINEAR APPROXIMATION TO σ_p IN TERMS OF σ_1 WHEN μ_1 IS FIXED

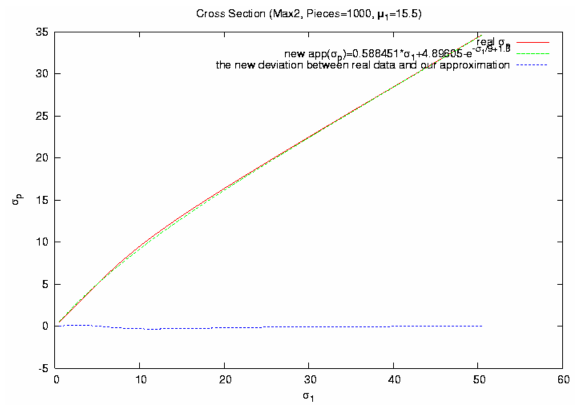


In this graph, we fixed μ_1 = 15.5 and used a linear function of σ_1 to approximate σ_p . The green line is our approximation, whereas the red curve is the real σ_p data. We can see the lower part of the curve dropped remarkably. So only using linear approximation does not produce a satisfactory result.

FIGURE 11

THE APPROXIMATION OF σ_p IN TERMS OF σ_1

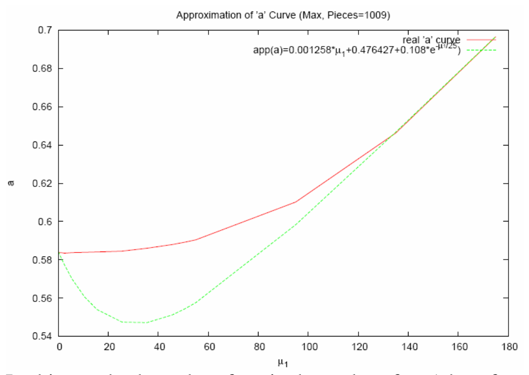
AFTER ADDING EXPONENTIAL COMPENSATION



In this figure, the red curve is the real σ_p data. The green curve is our approximation using linear function plus some exponential compensation. The blue curve is the error between the real data and our approximation, which is quite flat to the naked eye. Compared to Figure 10, we can see that our improved approximation method achieved a much better result. If we require more precise result, we could add more compensation to the method.

FIGURE 12

THE APPROXIMATION OF 'A' IN TERMS OF μ_1



In this graph, the red surface is the real perfect 'a' surface and the green surface is our approximation of 'a' in terms of μ_1 . There is a gap between the two curves in the area when μ_1 is relatively small. We could still use some compensation function to erase it. But in this case, the absolute value of error is acceptable. So we chose to keep the form of the approximation function simpler.

We applied the above procedure to different μ_1 values and found that σ_p could always be approximated by the function f(σ_1) in the form of f(σ_1) = a* σ_1 +b-exp(- σ_1 /d) and the parameters a, b, c and d vary regularly along the values of σ_1 . So we present the parameters a, b, c and d in term of μ_1 separately. Figure 12 shows the situation when we approximate the parameter a using a function of μ_1 .

After deriving all the four functions of approximating the parameters a, b, c and d in term of μ_1 , we substitute them into the perfect function σ_p = f(σ_1). Then we have the final approximation function of σ_p in terms of both μ_1 and σ_1 . We repeat the procedure to get the approximation function of μ_p , which is more precise than the result in Figure 9.

5. Use Case

After deriving the fundamental combination functions for runtime, we apply the BCC to some use case (Figure 13), which can be abstracted using our combination functions (Figure 14).

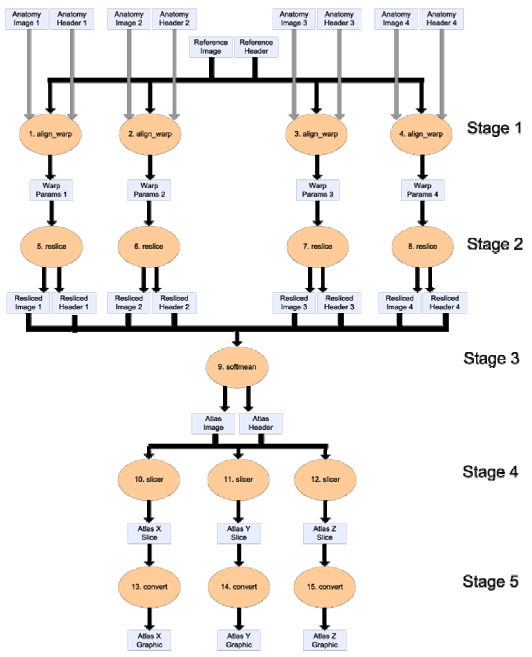
We use Agrajag results as the gold standard values against which to evaluate our BCC. Figure 16 shows the difference between the standard values and our approximation values. Through calculating the runtime of the whole workflow, we could:

(1) Do evaluations on the performance of the workflow;

(2) Provide measurement values for e-Scientists to make decisions on whether to use the workflow or not.

FIGURE 13

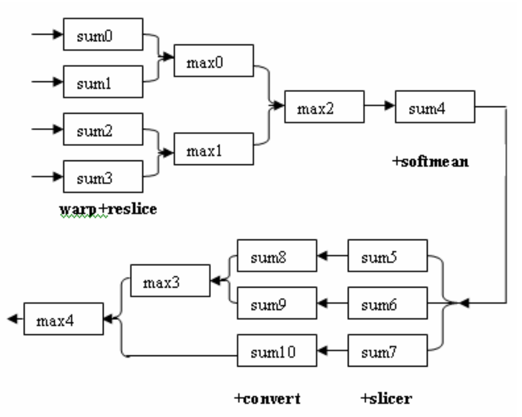
EVALUATION USE CASE



This is an example workflow for creating population-based "brain atlases", comprised of *procedures*, shown as orange ovals, and *data items* flowing between them, shown as rectangles. More details of the workflow can be found in http://twiki.ipaw.info/bin/view/Challenge/FirstProvenanceChallenge.

FIGURE 14

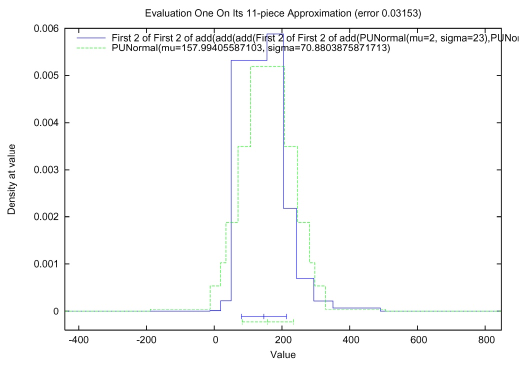
ABSTRACTED WORKFLOW USE CASE



This graph shows the abstract structure of the workflow using our fundamental combination functions. In this case, the workflow contains only the structures of 'Seq' and 'Para_All', so only 'sum' and 'max' are concerned.

FIGURE 15

APPROXIMATION OF RUNTIME



The graph presents the runtime of the workflow in Figure 13. In this graph, the blue curve shows the Agrajag values, namely, the standard values; while the green curve is our approximation values. Due to the low resolution, the difference is acceptable. If we raise the resolution, the advantage of the BCC will be obvious. It will take much less time to achieve the values.

6. Future Work

In Section 3~5, we gave the methodology and some experimental results of the BCC, and also applied it to certain use case. The next step is to complete the BCC by finishing the rest of the fundamental combination functions. Then we will find real data to do more evaluation. After that, we will consider extend the BCC by importing Log-Normal or other distributions to describe the values more precisely. We may also embed the extended BCC to some frameworks to enhance their functionalities of prediction and evaluation.

References

[1] Foster,I., Kesselman,C., The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers, 1999

[2] Smarr, L., Chapter 1. Grids in Context, in The Grid 2, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003

[3] Hillston, J. A Compositional Approach to Performance Modelling 1995

[4]http://www.statisticalengineering.com/central_limit_theorem.htm

[5]http://mathworld.wolfram.com/NormalSumDistribution.html