Uncertainty in the Cloud: an Angel-Daemon Approach to Modelling Performance *

A. Stewart(1) and J. Gabarro(2) and A. Keenan(1)

(1) School of EEECS, The Queen's University of Belfast, Northern Ireland.
(2) ALBCOM Research Group , Department of CS, Barcelona Tech, Spain
{a.stewart,a.keenan}@qub.ac.uk, gabarro@cs.upc.edu

Abstract. Uncertainty profiles are used to study the effects of contention within cloud and service-based environments. An uncertainty profile provides a qualitative description of an environment whose quality of service (QoS) may fluctuate unpredictably. Uncertain environments are modelled by strategic games with two agents; a daemon is used to represent overload and high resource contention; an angel is used to represent an idealised resource allocation situation with no underlying contention. Assessments of uncertainty profiles are useful in two ways: firstly, they provide a broad understanding of how environmental stress can effect an application's performance (and reliability); secondly, they allow the effects of introducing redundancy into a computation to be assessed.

Keywords. Uncertainty, Web-service, orchestration, ORC, cloud, virtualization, Amazon EC2, resource contention, performance, reliability, game theory.

1 Introduction

In 1961 John McCarthy proposed a vision of service-based computing:

> "If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility... "

The notion of service extends previous notions of programming through the addition of an interface through which users can access computing resources. There are many similarities between the service-based computation model and the established disciplines of sequential and parallel programming. Conventionally a state change is effected by an assignment statement $x := e$. The time take to execute this assignment is predictable and depends on the number of operations within e. In service-based computing state changes are effected through

* J. Gabarro is partially supported by funds from the Spanish Ministry for Economy and Competitiveness (MINECO) and the European Union (FEDER funds) under grant TIN2013-46181-C2-1-R (COMMAS) and also by SGR 2014:1137 (ALBCOM) from AGAUR, Generalitat de Catalunya.

service calls. A service (hosted in the cloud, or elsewhere) may have less predictable performance behaviour than a conventional imperative program – the QoS of service-based systems are considered in detail in [1]. Some of the factors influencing the performance of applications in the cloud are:

1. the type of hardware supplied by a provider to host a (virtual) machine;
2. the number of other (applications running as) VMs on a shared resource;
3. the behaviour of a hypervisor [9] supervising the execution of a set of VMs;
4. the nature of competing applications (e.g. web services, computationally intensive applications);

In this paper *uncertain execution environments* are specified in a *qualitative* way, using a two-player strategic game. One player (the daemon $\mathfrak{d}$) represents destructive stress; $\mathfrak{d}$ tries to maximise damage to an orchestration E by distributing a fixed degree of environmental stress (e.g. resource contention) over E's services. In contrast the angel player $\mathfrak{a}$ represents the self-healing capability of a system; the angel makes a move by allocating "benevolent conditions" to a fixed number of E's services (e.g. advantageous hardware allocation, no resource contention). The Nash equilibria of the resulting game provides a broad picture of how orchestrations react to mixed environmental stress.

The paper is organised as follows. In Section 2 an overview of the Orc language [7] and examples of performance variability in the cloud are given; an abstract game-theoretic (angel daemon) stress model is constructed. In Section 3 a cloud-based matrix multiplication orchestration is developed. The performance of matrix multiplication on a range of machine deployments is assessed using Nash equilibria in Section 4[1]. In Section 5 it is shown how game theory can be used to assess the effectiveness of adding redundancy to orchestrations. In Section 6 the applicability of the approach to other application areas is examined – for example variant forms of Angel-Daemon game could be used to analyse the effects of stress on orchestration communications.

2 Orc and a Model of Uncertain Cloud Environments

The language Orc [7] can be used to specify service-based computations and workflows [2]. A service s may fail to respond (i.e. it is *silent*) when it is called in an unreliable environment. A reliable service publishes a single result. In complex scenarios a service may call on further services and so may cause *side effects* elsewhere. Orc contains a number of inbuilt services: 0 is always silent whereas $1(x)$ always publishes its argument x. Two Orc expressions E and F can be combined using the following operators:

– Sequence $E > x > F(x)$: The orchestration E is evaluated: for each output v, published by E, an instance $F(v)$ is invoked. If E publishes the stream of values, $v_1, v_2, \ldots v_n$, then $E > x > F(x)$ publishes some interleaving

[1] All equiliria of a two person zero-sum game are identical – such assessments could also be computed by using linear programming.

of the set $\{F(v_1), F(v_2), \ldots, F(v_n)\}$. The abbreviation $E >> F$ is used in situations where F is independent of the publication value generated by E.
- Symmetric Parallelism $E \mid F$: The independent orchestrations E and F are executed in parallel; $E \mid F$ publishes *some* interleaving of the values published by E and F.
- Asymmetric parallelism $E(x) < x < F$: Orchestrations E and F are evaluated in parallel; E may become blocked by a dependency on x. The first result published by F is bound to x, the remainder of F's evaluation is terminated and evaluation of the blocked residue of E is resumed.

Example 1. Orchestration $Two(d) = (BBC(d)|CNN(d)) > x > Email(Bob, x)$ calls two news services in parallel on day d and sends the resulting publications, via an email service, to Bob. In contrast $One(d) = Email(Bob, x) < x < (BBC(d)|CNN(d))$ results in only one news summary for day d (the first available) being emailed to Bob. □

Uncertain cloud environments can be modelled in Orc. The infrastructure as a service (*IaaS*) cloud model allows users to control underlying hardware resources. Consider the following *IaaS* orchestration for multiplying two matrices, a and b:

$$P.provision(IMG) > MI > MI.deploy(MM) > MM_1 > MM_1(a, b)$$

Here a request is made to a provider P to supply a machine instance MI and configure it with an operating system image IMG; the machine instance is installed with a matrix multiply service MM; this service is then used to multiply the matrices a and b. The quality of service (QoS) realised by MM_1 depends on a number of environmental factors [1]. Typically *IaaS* clouds contain a variety of *machine types*. Table 1 shows some of the CPUs on 2012 AWS EC2.

Table 1: A Subset of CPUs available from AWS EC2 in 2012

Instance Type	Model	Speed (GHz)	L1 Cache	L2 Cache	L3 Cache
m1.small	AMD Opteron 2218 HE	2.6	2 x 64KB	2 x 1MB	N/A
m1.small	Intel Xeon E5420	2.66	4 x 64KB	2 x 6MB	N/A
m1.small	Intel Xeon E5507	2.26	4 x 64KB	4 x 256KB	4MB
c1.xlarge	Intel Xeon E5410 (×2)	2.333	4 x 64KB	2 x 6MB	N/A
c1.xlarge	Intel Xeon E5506 (×2)	2.133	4 x 64KB	4 x 256KB	4MB
cc1.4xlarge cg1.xlarge	Intel Xeon X5570 (×2)	2.933	4 x 64KB	4 x 256KB	8MB
cc2.8xlarge	Intel Xeon E5-2670 (×2)	2.6	8 x 64KB	8 x 256KB	20MB

In practice the performance of an application may depend critically on the amount of cache available on its execution platform. The performance of the MM_1 service may be influenced by the type of hardware supplied by the provider P in response to the service call $P.provision(IMG)$. Secondly it is important from a performance point of view that the installed service MM be tuned for the hardware supplied at run-time. In [8] a repository of tuned BLAS[2] implementations

[2] Basic Linear Algebra Subprograms (BLAS) are a library of low-level subroutines that perform common linear algebra operations

is made available in order to achieve tuning. The performance of $MM_1(a, b)$ on a shared multicore architecture may be critically influenced by the volume of traffic on the multicore bus which connects cores to on-chip memory. The performance of an orchestration E in a stressful environment (such as a cloud) can be modelled by associating a delay function, $\delta(s)$, with each underlying service s, $s \in \alpha(E)$[3]. Consider a model incorporating both *overdemand (o)* and *elasticity (e)*: *Overdemand* (demonic behaviour) may cause service degradation (e.g. multi-tenancy leads to memory contention); *Elasticity* (angelic behaviour) includes the allocation of the best type of resource and the deployment of extra resources to support a service, when needed. A tuple $(\delta(s), \delta_o(s), \delta_e(s), \delta_{o+e}(s))$ is a *stress model* [4] which specifies the performance delays associated with a service s:

- $\delta(s)$ is the delay of s in unstressed situations;
- $\delta_o(s)$ is the delay associated with s when it is subject to *overdemand*;
- $\delta_e(s)$ is the delay associated with s under *angelic* conditions;
- $\delta_{o+e}(s)$ is the delay when *overdemand* and *angelic* conditions interact.

The constraints: $\delta_e(s) < \delta(s) < \delta_o(s)$, $\delta_e(s) < \delta_{o+e}(s) < \delta_o(s)$ are assumed. A *stress model* for an orchestration E is a set $\mathcal{S}$ of underlying service stress models $\mathcal{S} = \{(\delta(s), \delta_o(s), \delta_e(s), \delta_{o+e}(s)) \mid s \in \alpha(E)\}$. Here orchestration performance in uncertain environments is assessed using a two-player game: one player, the daemon ($\mathfrak{d}$), has the potential to overload selected services and so increase delay (using the function δ_o). The other, the angel ($\mathfrak{a}$) has the potential to associate selected services with an idealised operating environment (δ_e). Stress-related performance delays for orchestrations are defined using two cost functions: $\Delta_{\max}(E)$ is the time taken for the generation of *all* publications of E and $\Delta_{\min}(E)$ is the time taken for the generation of the *first* publication. In the remainder of the paper we consider only pruning expressions of the form $E_3(x, y) < y < E_2 < x < E_1$ where the consumer E_3 is blocked until both producers E_1 and E_2 publish. Suppose that $[a, d]$ denotes the sets of services under the influence of $\mathfrak{a}$ and $\mathfrak{d}$, respectively. The delay associated with a service s is:

$$\Delta_{\min}(s)[a,d] = \Delta_{\max}(s)[a,d] = \begin{cases} \delta(s) & \text{if } s \notin a \wedge s \notin d. \\ \delta_o(s) & \text{if } s \in d \wedge s \notin a. \\ \delta_e(s) & \text{if } s \in a \wedge s \notin d. \\ \delta_{o+e}(s) & \text{if } s \in (a \cap d). \end{cases}$$

Orchestration delays are defined by:

$$\Delta_k(E_1 \mid E_2) = k\{\Delta_k(E_1), \Delta_k(E_2)\},\ \Delta_k(E_1 \gg E_2) = \Delta_k(E_1) + \Delta_k(E_2)$$
$$\Delta_k(s(x) < x < E) = \Delta_k(s) + \Delta_{\min}(E)$$
$$\Delta_k(s(x, y) < y < E_1 < x < E_2) = max\{\Delta_{\min}(E_1)\ ,\ \Delta_{\min}(E_2)\} + \Delta_k(s)$$

[3] $\alpha(E)$ denotes the set of services used in orchestration E – for example $\alpha(s_1(5)|s_2(8)) = \{s_1, s_2\}$, the two services used in the orchestration.

where $k \in \{\min, \max\}$. Thus $\Delta_{min}(E_1 \mid E_2) = min\{\Delta_{min}(E_1), \Delta_{min}(E_2)\}$, the time taken for the system $E_1 \mid E_2$ to generate its first publication. Uncertainty profiles (with cost functions) are used to capture formally the behaviour of orchestrations in stressed environments [3, 4]. The *uncertainty profile* $\mathcal{U} = \langle E, \mathcal{A}, \mathcal{D}, b_{\mathcal{A}}, b_{\mathcal{D}}, \Delta_{\max}\rangle$ specifies *qualitatively* a particular set of stress conditions for the orchestration E. Here $\mathcal{A} \cup \mathcal{D} \subseteq \alpha(E)$, $b_{\mathcal{A}} \leq \#\mathcal{A}$, $b_{\mathcal{D}} \leq \#\mathcal{D}$ and the cost function satisfies $\Delta_{\max}(E)[a, d] \geq 0$. Let $\alpha(E)$ denote the set of services used in E. In the profile:

- $\mathcal{A}$ and $\mathcal{D}$ denote the sets of services which can be influenced by $\mathfrak{a}$ and $\mathfrak{d}$, respectively. When stress can effect all services in E then $\mathcal{A} = \mathcal{D} = \alpha(E)$.
- Parameters $b_{\mathcal{A}}$ and $b_{\mathcal{D}}$ specify the number of services to suffer angelic and daemonic stress. For example, $(b_{\mathcal{A}}, b_{\mathcal{D}}) = (1, 1)$ exemplifies the weakest form of mixed stress while $(b_{\mathcal{A}}, b_{\mathcal{D}}) = (1, 2)$ is an unbalanced situation.
- The effect of stress on performance is measured by the cost function $\Delta_{\max}$.

Profile $\mathcal{U} = \langle E, \mathcal{A}, \mathcal{D}, b_{\mathcal{A}}, b_{\mathcal{D}}, \Delta_{\max}\rangle$ has an associated zero-sum *angel-daemon game* $\Gamma(\mathcal{U}) = \langle A_{\mathfrak{a}}, A_{\mathfrak{d}}, \Delta_{\max}\rangle$ with players $\mathfrak{a}$ (angel) and $\mathfrak{d}$ (daemon). Player $\mathfrak{a}$ selects $b_{\mathcal{A}}$ distinct stressed services from $\mathcal{A}$ giving the action set $A_{\mathfrak{a}} = \{a \subseteq \mathcal{A} \mid \#a = b_{\mathcal{A}}\}$. Player $\mathfrak{d}$ selects $b_{\mathcal{D}}$ distinct stressed services from $\mathcal{D}$ giving $A_{\mathfrak{d}} = \{d \subseteq \mathcal{D} \mid \#d = b_{\mathcal{D}}\}$. The set of combined actions $A = A_{\mathfrak{a}} \times A_{\mathfrak{d}}$ is called the *set of strategy profiles*. Given $\Gamma(\mathcal{U})$, player $\mathfrak{a}$ can "make a move" by selecting an action $a \in A_{\mathfrak{a}}$ (a is called a strategy). Likewise player $\mathfrak{d}$ can select an action $d \in A_{\mathfrak{d}}$. If both players select a strategy *independently* then the joint *strategy profile* is $s = (a, d)$. Players $\mathfrak{a}$ and $\mathfrak{d}$ have costs $\Delta_{\max}(E)[a, d]$ and $-\Delta_{\max}(E)[a, d]$, respectively. The angel player $\mathfrak{a}$ wishes to minimise an orchestration's cost delay whereas the daemon $\mathfrak{d}$ wishes to maximise it. Mixed strategies for players $\mathfrak{a}$ and $\mathfrak{d}$ are probability distributions $\alpha : A_{\mathfrak{a}} \to [0, 1]$ and $\beta : A_{\mathfrak{d}} \to [0, 1]$, respectively. A *mixed strategy profile* is a tuple (α, β) such that $\Delta_{\max}(E)[\alpha, \beta] = \sum_{(a,d)\in A_{\mathfrak{a}}\times A_{\mathfrak{d}}} \alpha(a)\Delta_{\max}(E)[a, d]\beta(d)$. Let $\Delta_{\mathfrak{a}}$ and $\Delta_{\mathfrak{d}}$ denote the set of mixed strategies for players $\mathfrak{a}$ and $\mathfrak{d}$, respectively. A pure strategy profile (a, d) is a special case of a mixed strategy profile (α, β) in which $\alpha(a) = 1$ and $\beta(d) = 1$. A mixed strategy profile (α, β) is a *Nash equilibrium* if for any $\alpha' \in \Delta_{\mathfrak{a}}$, $\Delta_{\max}(E)[\alpha, \beta] \leq \Delta_{\max}(E)[\alpha', \beta]$ and for any $\beta' \in \Delta_{\mathfrak{d}}$, $\Delta_{\max}(E)[\alpha, \beta] \geq \Delta_{\max}(E)[\alpha, \beta']$. A pure Nash equilibrium, PNE, is a Nash equilibrium (a, d) where a and d are pure strategies. The value of the zero-sum game $\Gamma(\mathcal{U})$ associated with the uncertainty profile $\mathcal{U}$ is denoted by $\nu(\mathcal{U})$ is $\nu(\mathcal{U}) = \min_{\alpha\in\Delta_{\mathfrak{a}}} \max_{\beta\in\Delta_{\mathfrak{d}}} \Delta_{\max}(E)[\alpha, \beta] = \max_{\beta\in\Delta_{\mathfrak{d}}} \min_{\alpha\in\Delta_{\mathfrak{a}}} \Delta_{\max}(E)[\alpha, \beta]$. Strategy (α, β) is a Nash equilibrium iff $\Delta_{\max}(E)[\alpha, \beta] = \nu(\mathcal{U})$.

3 Matrix Multiplication in the Cloud

A conventional block matrix multiplication (BMM) of an $p \times r$ block matrix A and a $r \times q$ block matrix B can be defined using the r-way partition: $C_{ij} = \sum_{k=1}^{r} A_{ik}B_{kj}$, $1 \leq i \leq p$, $1 \leq j \leq q$, where A_{ik} and B_{kj} denote the blocks of A and B. The case $p = q = r = 2$ is shown.

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} AB = \begin{bmatrix} (A_{11}B_{11} + A_{12}B_{21}) & (A_{11}B_{12} + A_{12}B_{22}) \\ (A_{21}B_{11} + A_{22}B_{21}) & (A_{21}B_{12} + A_{22}B_{22}) \end{bmatrix}$$

Suppose that the services *MM* and *MA*, for multiplying and adding small and medium sized matrices, are deployed in the cloud. For example, the Amazon EC2 m1.small instance has a 1.7GB RAM capacity (enough to accommodate three 64-bit precision 8000×8000 matrices) while the EC2 c1.xlarge instance type has a 7GB RAM capacity (enough to accommodate three 16000×16000 matrices). Matrices of larger size can be multiplied together by constructing a parallel BMM orchestration which generates *block* matrix-vector and dot-product subtasks.

$$\begin{aligned} &BMM_{2\times2}([a,b,c,d],[e,f,g,h]) = \\ &\quad 1([w,x,y,z]) < w < DP([a,b],[e,g]) < x < DP([a,b],[f,h]) \\ &\qquad\qquad < y < DP([c,d],[e,g]) < z < DP([c,d],[f,h]) \end{aligned}$$

Block dot products may be implemented either sequentially or in parallel:

$$SeqDP_{2\times2}([a,b],[c,d]) = MM(a,c) > m_1 > MM(b,d) > m_2 > MA(m_1,m_2)$$
$$DP_{2\times2}([a,b],[c,d]) = MA(m_1,m_2) < m_1 < MM(a,c) < m_2 < MM(b,d)$$

Refinement to an IaaS Orchestration. $BMM_{2\times2}$ can be refined to an orchestration which operates in the infrastructure as a cloud model ($IaaS$); here cloud hardware resources can be provisioned and managed *explicitly*. The IaaS-level orchestration below has in its argument list the name of a cloud provider, P, an operating system image, *IMG*, as well as the services *MM* and *MA*. The subsidiary orchestration DP_I uses P and *IMG* to provision two machines for each dot-product.

$$\begin{aligned} &BMM_I([[A_{11},A_{12}],[A_{21},A_{22}],[B_{11},B_{21}],[B_{12},B_{22}]],P,IMG,MM,MA) = \\ &\quad 1([[C_{11},C_{21}],[C_{12},C_{22}]]) \\ &\qquad < C_{11} < DP_I([A_{11},A_{12}],[B_{11},B_{21}],P,IMG,MM,MA) \\ &\qquad < C_{12} < DP_I([A_{11},A_{12}],[B_{12},B_{22}],P,IMG,MM,MA) \\ &\qquad < C_{21} < DP_I([A_{21},A_{22}],[B_{11},B_{21}],P,IMG,MM,MA) \\ &\qquad < C_{22} < DP_I([A_{21},A_{22}],[B_{12},B_{22}],P,IMG,MM,MA) \\ &DP_I([A_1,A_2],[B_1,B_2],P,IMG,MM,MA) = \\ &\quad M_1.deploy(MA) > MA_1 > MA_1(x,y) \\ &\qquad < x < M_1.deploy(MM) > MM_1 > MM_1(A_1,B_1) \\ &\qquad < y < M_2.deploy(MM) > MM_2 > MM_2(A_2,B_2) \\ &\qquad\quad < M_1 < P.provision(IMG) < M_2 < P.provision(IMG) \end{aligned}$$

A Stress Model for BMM. Performance results for executing *BMM* on clusters of Amazon EC2 c1.xlarge instances (8 CPU cores per instance) are shown in Table 2:

Table 2: Average, Minimum and Maximum Times (in seconds) for *BMM*

Matrix Size	Block Size	Instances Used	Avg	Min	Max
8000	8000	1	41 (m)	24	76
16000	16000	1	297.22	171	408
	8000	4	151.95	94	206
	8000	8	103.55	77	155

Times are calculated using 20 separate tests. Execution time is measured remotely from a client and includes internet latency. A stress model for *BMM* is constructed by mapping $\delta_e(MM)$ and $\delta_o(MM)$ onto the minimum and maximum execution times for MM, respectively. The data for the one instance 8000×8000 experiment is used to build a *qualitative* model[4]. Speed-up predictions made by the uncertainty model are upper-bounds on actual speed-ups [8] since latency is not taken into account. The following stress model $\mathcal{S}$ results:

$$\delta(MA) = \delta_{o+e}(MA) = a,\ \delta(MM) = \delta_{o+e}(MM) = m,$$
$$\delta_o(MA) = 2 * a,\ \delta_o(MM) = 2 * m,\ \delta_e(MA) = 0.5 * a,\ \delta_e(MM) = 0.5 * m$$

4 Assessing BMM Orchestrations under Stress

The *IaaS* model allows application developers to control directly the degree of parallelism employed by a cloud implementation. Three possible *IaaS* deployment configurations for BMM are considered below:

- *single machine BMM*: The performance is *estimated* by the performance of a intra-machine deployment (sequential implementation) of $BMM_{2\times2}$.
- *dot product inter-machine virtualisation* (BMM_{SeqDP}): here a separate machine instance is allocated to each of the 4 dot products in BMM.
- *fully parallel inter-machine virtualisation* (BMM_I): here a separate machine instance is allocated to each of the 8 matrix multiplication services.

BLAS Routines and Intra-machine Virtualization. Matrix multiplication is implemented on a single machine instance using BLAS library calls. Performance is modelled using a *uniform stress model* where all services (on a single core) are subject to the same level of stress. The performance of sequential dot product in an environment with a stress level l, $l \in \{o, e, o+e\}$ is estimated by $\Delta_l(SeqDP_{2\times2}) = 2\delta_l(MM) + \delta_l(MA)$. The performance of sequential $BMM_{2\times2}$ under mixed stress is estimated by $\Delta_{o+e}(BMM_{2\times2}) = 4\Delta_{o+e}(SeqDP_{2\times2}) = 4(2m + a)$ Since $m >> a$ then $\Delta_{o+e}(BMM_{2\times2}) \approx 8m$ (roughly in line with the experimental data for sequential matrix multiplication on 8000×8000 and 16000×16000 data – see Table 2).

[4] The cost of matrix multiplication etc depend on the problem size. However, in order to simplify the analysis a fixed problem size is used.

Inter-machine Virtualization. A uniform stress model does not capture the uncertain nature of service-based environments where deployment may take place on independent machine instances. Orchestration BMM_{SeqDP} has independent machine instances allocated to each dot product:

$$\begin{aligned} BMM_{SeqDP}([a,b,c,d],[e,f,g,h]) = \\ 1([w,x,y,z]) < w < SeqDP_1([a,b],[e,g]) < x < SeqDP_2([a,b],[f,h]) \\ < y < SeqDP_3([c,d],[e,g]) < z < SeqDP_4([c,d],[f,h]) \end{aligned}$$

The profile $\mathcal{U} = \langle BMM_{SeqDP}, S, S, 1, 1, \Delta_{\max}\rangle$ where $\mathcal{S} = \{SeqDP_1, \ldots, SeqDP_4\}$ models BMM_{SeqDP} under moderate balanced stress and gives rise to the game:

$\mathfrak{a}$ \ $\mathfrak{d}$	DP_1	DP_2	DP_3	DP_4
DP_1	$2m+a$	$4m+2a$	$4m+2a$	$4m+2a$
DP_2	$4m+2a$	$2m+a$	$4m+2a$	$4m+2a$
DP_3	$4m+2a$	$4m+2a$	$2m+a$	$4m+2a$
DP_4	$4m+2a$	$4m+2a$	$4m+2a$	$2m+a$

The strategy $\alpha = \beta = (1/4, 1/4, 1/4, 1/4)$ is an equilibrium with delay $\Delta(\alpha,\beta) = \sum_{i,j} \alpha(DP_i)\beta(DP_j)\Delta(DP_i, DP_j) = 7(2m+a)/4 \approx 7m/2$. Table 2 shows that multiplication of matrices of size 16000×16000, takes $151.95 \approx 3.7m$ seconds.

Fully Parallel IaaS Deployment. In order to achieve high performance all MM instances in $BMM_{2\times2}$ are called in parallel (using $DP_{2\times2}$). The orchestration BMM_I has eight (parallel) instances of MM and four instances of MA. Thus $\mathcal{S} = \{MM_1, \ldots, MM_8, MA_1, \ldots, MA_4\}$ is the set of services under the influence of stress. Profile $\mathcal{U} = \langle BMM_I, \mathcal{S}, \mathcal{S}, 1, 1, \Delta_{\max}\rangle$ provides a model of the behaviour of BMM_I in a mixed stress environment (where both angel and daemon have the capacity to influence a single service) and gives rise to the associated game

$\mathfrak{a}$ \ $\mathfrak{d}$	MM_j	MA_l
MM_i	$(m+a) \triangleleft (i=j) \triangleright (2m+a)$	$m+2a$
MA_k	$2m+a/2$	$(m+a) \triangleleft (k=l) \triangleright (m+2a)$

$i, j \in \{1, \ldots, 8\}$ and $k, l \in \{1, \ldots, 4\}$

Here the notation $P \triangleleft b \triangleright Q$ (denoting '$P\ if\ b\ else\ Q$") is used to provide a compact description. If both players choose MM_1 then BMM_I will have performance $m+a$; however if they choose different MM instances then performance degrades to $2m + a$ (a parallel computation is only as fast as its slowest component). When $m \geq 4a$, $\alpha = \beta = (1/8, 1/8, 1/8, 1/8, 1/8, 1/8, 1/8, 1/8, 0, 0, 0, 0)$ is a mixed equilibrium such that $\Delta(\alpha,\beta) = \frac{15}{8}m + a$. Thus mixed stress is predicted to degrade the performance of BMM_I from an optimium $m + a$ to $15m/8 + a$. In this case there is a discrepancy between experimental data ($\approx 2.5m$) and the game theory predication ($\approx \frac{15}{8}m$). However, the latter approach provides a much better performance estimate than a uniform stress model ($\approx m$).

5 Redundancy and Increased Stress Levels

In practice *BMM* orchestration deployments with 64 or more machine instances may have reliability issues due either to slow or non-responsive services [6]. The $\mathfrak{a}/\mathfrak{d}$ approach can be applied to reason about the use of redundancy to improve orchestration resilience. Service $DP_{2\times2}$ can be enhanced by the addition of duplicate multiplication services:

$$\begin{aligned} &\mathit{rdntMM\_DP}_{2\times2}([a,b],[c,d]) = \\ &\quad MA(m_1,m_2) < m_1 < \big(1(m) < m < (MM_1(a,c) \mid MM_2(a,c))\big) \\ &\qquad\qquad < m_2 < \big(1(m) < m < (MM_3(b,d)) \mid MM_4(b,d))\big) \end{aligned}$$

Here MM_1, MM_2, MM_3, MM_4 are independent services. A mixed-stress profile for *rdntMM_DP* is $\mathcal{U}_{\mathit{rdntMM\_DP}} = \langle \mathit{rdnt\_DP}_{2\times2}\mathcal{S}, \mathcal{S}, 1, 1, \Delta_{\max}\rangle$ where $\mathcal{S} = \{MA, MM_1, MM_2, MM_3, MM_4\}$. The associated $\mathfrak{a}/\mathfrak{d}$-game has (MA, MA) as PNE with valuation $m + a$. Thus, it is predicted that adding redundancy improves performance of dot product from $3m/2 + a$ to $m + a$. Four *rdntMM_DP* can be incorporated within $BMM_{2\times2}$ in order to improve the overall QoS. This situation is assessed using the profile $\mathcal{U} = \langle \mathit{rdntMM\_BMM}_{2\times2}, \mathcal{S}, \mathcal{S}, 1, 1, \Delta_{\max}\rangle$ where $\mathcal{S} = \{MA_1, \ldots, MA_4, MM_1, \ldots, MM_{16}\}$. The resulting $\mathfrak{a}/\mathfrak{d}$-game has (MA, MA) as a PNE with valuation $m + a$ (compared to the estimated $15m/8 + a$ for the normal implementation).

Dot product with in-built redundancy (above) can be analysed in a scenario with increased stress where the daemon influences two services whereas the angel can only moderate one. The situation is captured by the profile $\mathcal{U} = \langle \mathit{rdntMM\_DP}_{2\times2}, \mathcal{S}, \mathcal{S}, 1, 2, \Delta_{\max}\rangle$, $\mathcal{S} = \{MM_1, \ldots, MM_4, MA\}$. The associated game has no PNE. However mixed equilibria have valuations $3/2m + a$. Thus, additional stress causes dot product (with redundancy) to deteriorate from $m + a$ to $3/2m + a$. The behaviour of the full multiplication orchestration $\mathit{rdntMM\_BMM}_{2\times2}$ under increased stress can be analysed in a similar way using the profile $\mathcal{U} = \langle \mathit{rdntMM\_BMM}_{2\times2}, \mathcal{S}, \mathcal{S}, 1, 2, \Delta_{\max}\rangle$. Provided that $m \geq 4a$ the value of the $\mathfrak{a}/\mathfrak{d}$-game is $15m/8 + a$. Thus, additional stress is predicted to cause an approximate doubling in the execution time of *rdntMM_BMM*.

6 Discussion

There are well established theories for estimating the performance of sequential and parallel computations with respect to the number of operations that are executed for a given size of input. Analysing performance in service-based environments is much more complex. A conventional orchestration cost model captures behaviour in favourable operating conditions. More generally, the behaviour of an orchestration is dependent on the current level of environmental stress and the resilience of underlying services. In this paper uncertainty profiles are used to model the competitive circumstances that arise when services are subject to the effects of both overdemand and elasticity. The model $\mathfrak{a}/\mathfrak{d}$-model provides an extra layer of understanding about the evaluation of complex orchestrations.

There is a reasonable correlation between the predictions made by the $\mathfrak{a}/\mathfrak{d}$-model and experimental data. It is important to remember that uncertainty profiles are *qualitative* descriptions of evaluation environments; $\mathfrak{a}/\mathfrak{d}$-games provide a broad picture of how stress affects orchestration behaviour. Our aim is to provide a framework in which designers can analyse the effects of resource contention on services and orchestrations (rather than having to rely on a trial-and-error approach). In the paper we demonstrate how $\mathfrak{a}/\mathfrak{d}$ performance parameters can be constructed from experimental data. Perhaps the usefulness of the model can be seen most clearly when analysing the stress resilience capabilities of a number of different forms of a workflow.

It is not clear how the $\mathfrak{a}/\mathfrak{d}$ approach *scales* with orchestration size. In general it may be difficult to calculate Nash equilibria for large irregular orchestrations. However, it should be noted that there are practical techniques for finding mixed equilibria of large games [5]. In this paper attention has been focused on the effect that resource contention can have on machine and orchestration performance.

References

1. Albert Benveniste, Claude Jard, Ajay Kattepur, Sidney Rosario, and John A. Thywissen. Qos-aware management of monotonic service orchestrations. *Formal Methods in System Design*, 44(1):1–43, 2014.
2. William R. Cook, Sourabh Patwardhan, and Jayadev Misra. Workflow patterns in Orc. In Paolo Ciancarini and Herbert Wiklicky, editors, *Coordination Models and Languages, 8th International Conference, COORDINATION 2006, Bologna, Italy, June 14-16, Proceedings*, volume 4038 of *LNCS*, pages 82–96. Springer, Berlin, 2006.
3. J. Gabarro, M. Serna, and A. Stewart. Web services and *incerta spiriti*: A game theoretic approach to uncertainty. In W. Liu, editor, *ECSQARU 2011, Belfast, UK, June 29-July 1*, volume 6717 of *LNCS*, pages 651–662, Berlin, 2011. Springer-Verlag.
4. Joaquim Gabarro, Maria Serna, , and Alan Stewart. Analysing web-orchestrations under stress using uncertainty profiles. *Comput. J.*, 57(11):1591–1615, 2014.
5. Albert Xin Jiang, Manish Jain, and Milind Tambe. Computational game theory for security and sustainability. *JIP*, 22(2):176–185, 2014.
6. Antony Keenan. Orchestrating Hight Performance Services: Theory and Practice, PhD Thesis, 2014.
7. Jayadev Misra and William R. Cook. Computation orchestration: A asis for wide-area computing. *Software and System Modeling*, 6(1):83–110, 2007.
8. Terence Harmer Anthony Keenan Alan Stewart Peter Wright, Yih Leong Sun and Ronald Perrott. A constraints-based resource discovery model for multi-provider cloud environments,. *Journal of Cloud Computing: Advances, Systems and Applications*, 1:1–14, 2012.
9. Carl A. Waldspurger and Mendel Rosenblum. I/O virtualization. *Commun. ACM*, 55(1):66–73, 2012.