Fault-tolerant virtual cluster experiments on federated sites using BonFIRE

Grupo:

Edgar Oliveira, E8385
Pedro Jesus, M6369

O Problema:

A falha dos sites na Cloud e a variabilidade da performance das máquinas virtuais (VMs) neste ambiente.

- Exemplo:

  No verão de 2012, dois dos maiores fornecedores de serviços cloud sofreram interrupções no serviço em algumas áreas. Como consequência, serviços dos seus clientes estiveram total ou parcialmente parados.

1. Introdução

Tem-se observado uma variação na performance na Amazon EC2, mesmo em dois níveis diferentes de performance para a mesma configuração da mesma infraestrutura virtual mas medidas em alturas diferentes.

1. Introdução (cont.)

Neste caso, para manter a qualidade do serviço, uma possível estratégia é explorar a elasticidade da Cloud para se auto adaptar a esta variabilidade indesejada.

Significa isto usar um Indicador Chave da performance da Aplicação (KAI) para activar o alargamento da infraestrutura virtual que o suporta quando necessário ou para o reduzir quando em repouso, para evitar custos desnecessários.

1. Introdução (cont.)

Para verificar o desenho desta arquitetura, esta foi testada em três experiências:

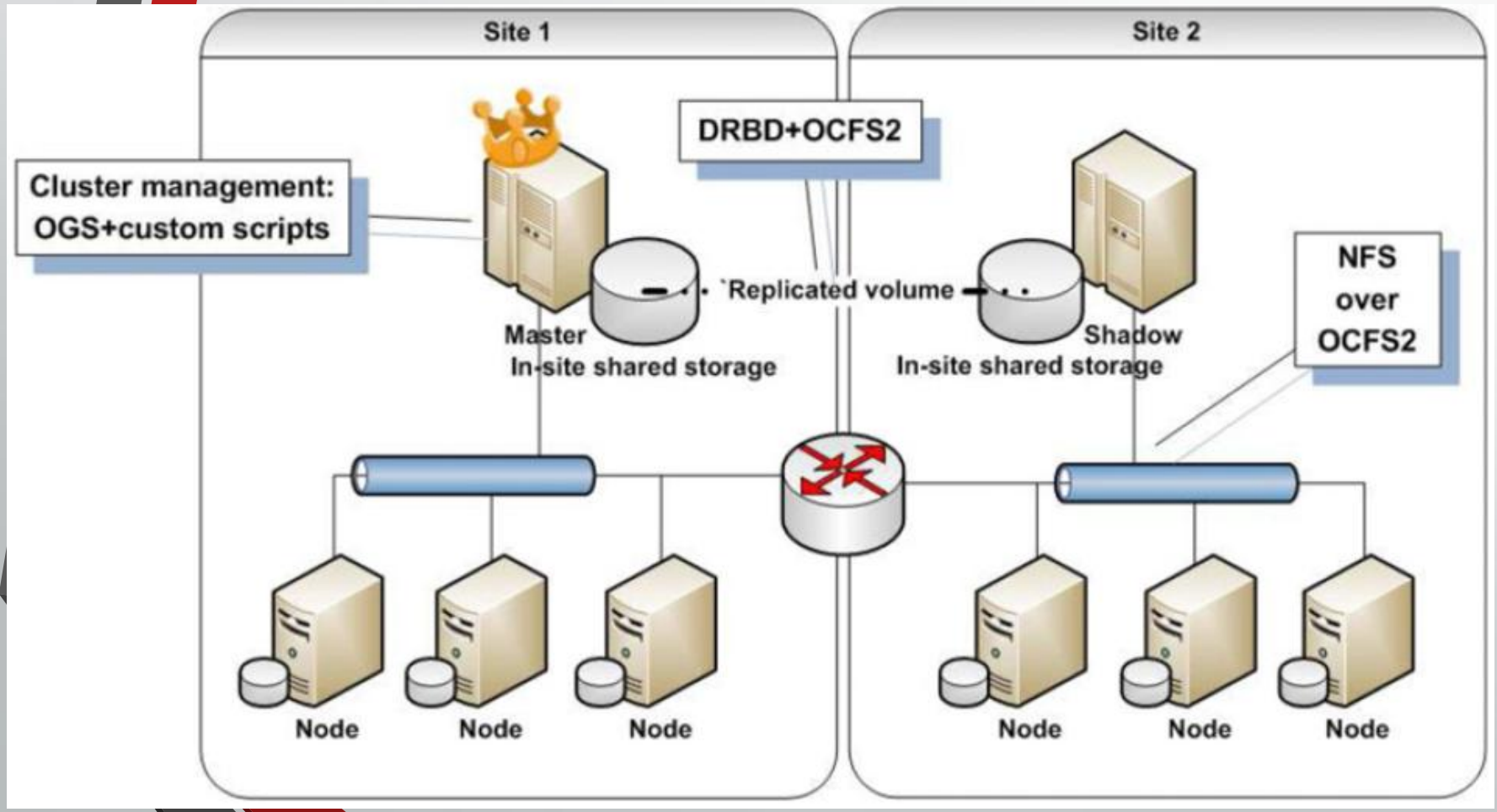
- Execução da aplicação numa configuração de múltiplos sites
- Experimentação sobre Objetivos Específicos de Deadline onde o Elasticity Engine toma decisões sobre o alargamento do Cluster com novas VMs para atingir o Deadline.

1. Introdução (cont.)

- E um teste de Tolerância a Falhas onde se simula a perda de uma parte do cluster virtual, restaurando a performance da aplicação no Site da Cloud sobrevivente, usando mecanismos de recuperação e regras de elasticidade, sem interrupção do serviço.

2. Materiais e métodos

[2.1] Arquitectura de Cluster Virtual



A Arquitetura de Cluster Virtual (VC) contempla:

- dois nodos principais ("Master" e "Shadow")
- dois conjuntos de nodos de computação (CE); cada conjunto está associado a um dos nodos principais.

Portanto, o VC é separado em duas partições que podem ser implantado em localizações diferentes.

[2.1] Arquitetura de Cluster Virtual (cont.)

Para o construir, duas imagens de VM têm de ser configuradas:

- uma para os nodos principais (master e shadow);
- outra para os nodos.

As dependências entre nodos, a localização de cada partição, e outros requerimentos técnicos são automaticamente autoconfigurados durante a instalação final.

[2.1] Arquitectura de Cluster Virtual (cont.)

- O Nodo "Master" tem quatro tarefas:
  - Servidor de dados para os CEs que estão localizados no mesmo Site da Cloud.
  - Controlador da queue
  - Gestor de aplicações
  - E configuração do Cluster
- O Nodo "Shadow" tem três tarefas:
  - Servidor de dados para os seus clientes.
  - Controlador da queue de backup.
  - Gestor das aplicações de backup.
- Os Nodos de computação (CEs) têm apenas uma tarefa:
  - Execução de trabalhos.

[2.1] Arquitectura de Cluster Virtual (cont.)

Os quatro componentes que o VC tem de ter para executar as tarefas descritas atrás:

1 - Recursos de armazenamento partilhados:

É necessário para controlar e sincronizar o sistema de 'queue', e para armazenar todos os dados dos processos em execução.

É também incluído um volume vazio em cada um dos Nodos principais (master e shadow). O conteúdo de ambos os volumes é replicado e sincronizado. A sincronização inicial pode implicar uma elevada carga no setup do VC. Para diminuir essa carga, os volumes partilhados são criados previamente em sites BonFIRE como DATABLOCKs e preenchidos de zeros.

[2.1] Arquitetura de Cluster Virtual (cont.)

2 - Sistema de Queue:

Para distribuir tarefas paralelas entres os nodos é usado o "Open Grid Scheduler" (OGS). Os nodos master e shadow são configurados como master daemon e shadow daemon no OGS, respectivamente. Ambos são submit hosts e administration hosts.

Apenas os CEs são execution hosts. Independentemente da sua partição podem comunicar com o master (ou shadow quando o master não está disponível) para operações de queue ou distribuição de tarefas.

[2.1] Arquitetura de Cluster Virtual (cont.)

3- Autoconfiguração:

Intervenção externa indesejável depois de submeter o pedido de setup do VC.

Por isso o cluster autoconfigura-se. O master detecta as máquinas iniciais no cluster e configura todos os componentes. Também conecta ao shadow usando SSH para fazer a sua configuração.

[2.1] Arquitetura de Cluster Virtual (cont.)

4 - Software da aplicação:

Tem de ser uma aplicação que pode ser dividida em várias tarefas e que pode informar o VC sobre a sua performance, usando um indicador, KAI.

Também para ser completamente autónomo, se estas tarefas são controladas por uma aplicação mestre, deve ser executada no cluster e deve ser tolerante a falhas, i.e., deve recuperar de falhas no restart ou ser cluster ready.

[2.1] Arquitetura de Cluster Virtual (cont.)

Baseado nesta arquitetura geral, podem-se implementar três configurações diferentes numa Cloud:

- Single-site Virtual Cluster: O caso mais simples. O cluster é instalado numa única localização física e inclui apenas os Nodos Master e de computação.

- Distributed Virtual Cluster: Apenas os Nodos Master e de computação são lançados, mas as últimas VM são distribuídas entre dois sites (ou fornecedores de Cloud). Pode ser usado num setup hibrido de Cloud, combinando Clouds públicas e privadas, ou para ultrapassar a limitação de recursos nos fornecedores.

[2.1] Arquitetura de Cluster Virtual (cont.)

Baseado nesta arquitectura geral, podem-se implementar três configurações diferentes numa Cloud:

- Fault Tolerant Virtual Cluster: o cluster é implementado em dois sites para ser tolerante a falhas de sites.

  O master está num site, o shadow em outro, e os nodos de computação estão distribuídos entre os dois.
  Nesta configuração, se um site falha, a restante estrutura do cluster pode recuperar e aplicar regras de elasticidade, alargando-se para fornecer resultados para o utilizador em tempo útil.

[2.1] Arquitectura de Cluster Virtual (cont.)

A configuração dos CEs é completamente assíncrona, podendo assim ser adicionados a qualquer altura.

Esta arquitetura de cluster suporta ainda dois tipos de falhas:

- Perda de CEs: neste caso as tarefas simplesmente têm de ser redistribuídas para outro nodo.

- Falha completa de uma localização: quando a localização que falha é host do master, o shadow pode ficar com o papel de master e continuar a execução do cluster.

[2.2] Elasticity Engine

- O algoritmo de elasticidade monitoriza a performance da aplicação periodicamente e gere a adição ou remoção de CEs para chegar ao Tempo Limite Desejado (DTL) para um Indicador Chave da Aplicação (KAI).
  - Tempo Limite Desejado (DTL)
  - Indicador Chave da Aplicação (KAI)
  - performance do cluster (CP)
  - performance estimada (EP)
  - (LOW_PERFORMANCE_INTERVAL)
- O numero de cores é calculado como um múltiplo do número de cores em atividade, segundo a equação:
  - coresNeeded = min(MAX_CORES, (EP/CP – 1) ∗ totalCores)

[2.2] Elasticity Engine (cont.)

- Quando a performance do cluster (CP) é maior que a performance estimada (EP), o DTL pode ser atingido com um número menor de nodos, reduzindo o custo de execução.

- Para calcular o número de cores a remover é usada a seguinte expressão:
  - coresNeeded = ((1 + MARGIN) ∗ EP/CP – 1) ∗ totalCores

Algorithm 1 Elasticity algorithm

Require: *DTL* = Desired Time Limit
Require: *KAI* = Key Application Indicator
{Elasticity algorithm}
repeat
sleep(ELASTICITY PERIOD)
totalCores ← number of cores of client nodes at master and shadow locations
elapsedTime ← elapsed time
for all nodes **do**
$rate_i$ ← Application Indicator rate in node *i*
$done_i$ ← Application Indicator done in node *i*
end for
$CP = \sum rate_i$ {Cluster Application Indicator rate}
$CT = \sum done_i$ {Cluster Application Indicator done}
$EP = (KAI - CT)/(DTL - elapsedTime)$ {Number of remaining KAI/remaining time= Required rate}
coresNeeded = 0
if $CP < EP$ **then**
coresNeeded = min(MAX_CORES,(EP/CP - 1)*totalCores)
end if
if $CP > EP$ **then**
coresNeeded = ((1+MARGIN)*EP/CP - 1)*totalCores
end if
if *coresNeeded* > 0 for LOW_PERFORMANCE_INTERVAL time **then**
DeployVMs(coresNeeded)
sleep(DEPLOYMENT_PERIOD)
end if
if *coresNeeded* < 0 **then**
CancelVMs(coresNeeded)
sleep(UNDEPLOYMENT_PERIOD)
end if
until (*KAI* == *CT*)

[2.2] Elasticity Engine(cont.)

O algoritmo requer calibração destes parâmetros para cada fornecedor de Cloud, um monitor específico para a aplicação para esta recolher dados sobre a sua performance e um mecanismo para arrancar VMs novas a partir do cluster sem input externo.

Na experiência a seguir o Elasticity Engine usa as "features" do BonFIRE para oferecer às VMs credenciais para aceder à API do BonFIRE, assim o master node pode gerir recursos sem intervenção do utilizador.

[2.3] Infraestrutura BonFIRE

- BonFIRE é um projecto para fornecer um serviço de teste em multi-cloud para a comunidade da Internet dos serviços. Inside em vários servidores de cloud na europa, França, Ingaterra, Alemanha, Belgica.
- Cada um destes especifíca a sua máquina virtual.
- Para instalar o cluster virtual foi utilizado o Experiment Manager da arquitetura BonFIRE usando um script de definição JSON.

[2.3] Infra-estrutura BonFIRE (cont.)

- O Experiment Manager interpreta o input JSON e pede as diferentes VMs aos sites Cloud finais.
- O Experiment Manager devolve um Experiment ID que permite mais tarde recolher informação sobre o seu estado ou de cada um dos componentes. Após iniciado o cluster os elementos da experiência podem ser mudados (adicionar ou remover VMs) usando o BonFIRE Resource Manager.
- Este pedido pode ser feito do exterior ou pelas próprias VMs que o compõem.
- O mesmo mecanismo explicado no Elasticity Manager anteriormente.

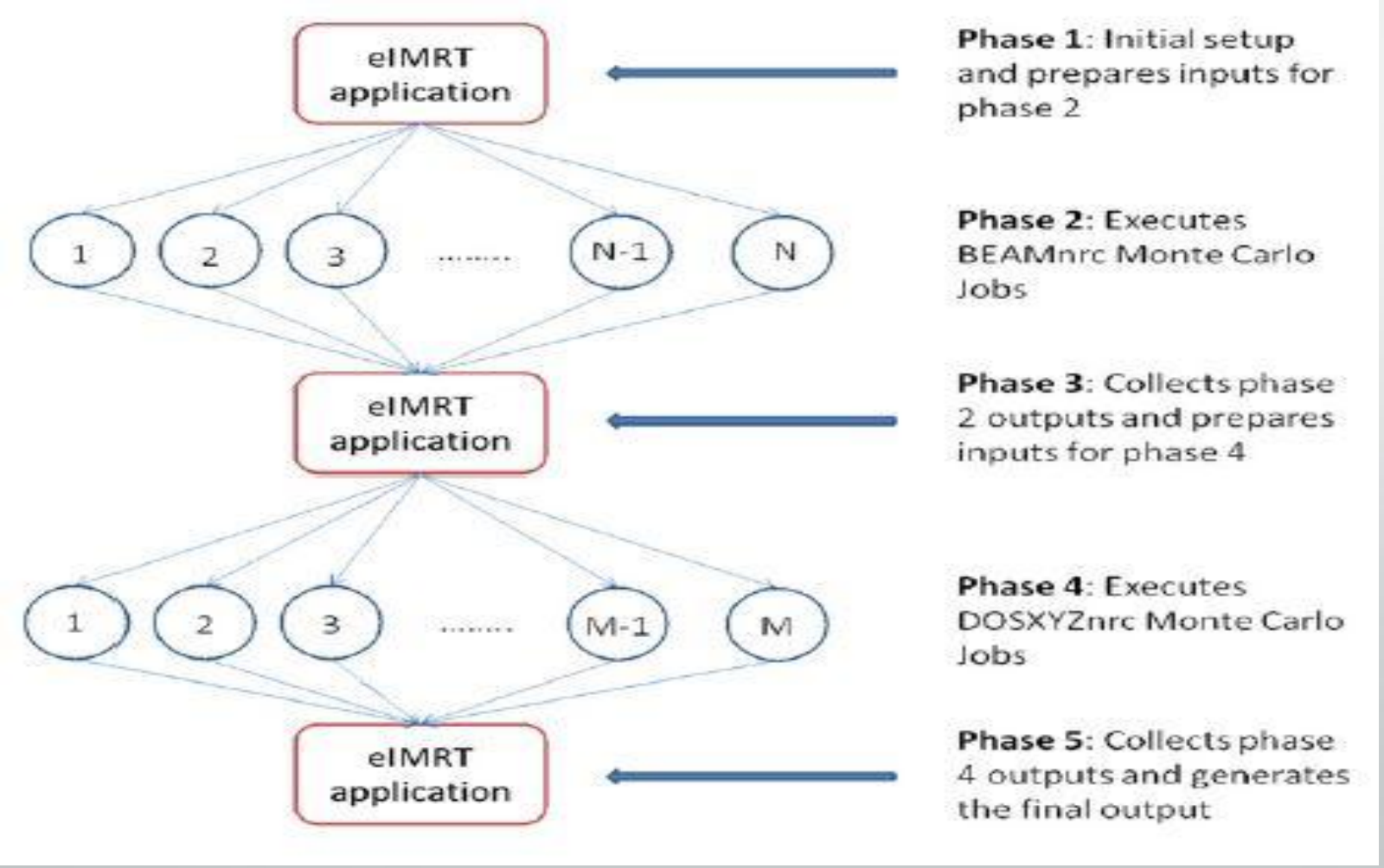
[2.4] Plataforma eIMRT

- É uma infra-estrutura de computação remota para o fornecimento de ferramentas computacionais intensivas para o planeamento de radioterapia utilizando o paradigma "software como um serviço".

- Consiste na verificação de tratamentos por radioterapia e recalculá-los utilizando métodos de "Monte Carlo" , comparando os resultados obtidos com algoritmos rápidos mas com menor precisão.

- Métodos de "Monte Carlo" consistem na combinação dos pacotes de software BEAMnrc e DOSXYZnrc.

O processo de verificação é baseado num conjunto de 5 etapas por forma a calcular as doses administradas ao paciente, como é possível visualizar no esquema abaixo.



Etapa 1

- Leitura dos ficheiros de input.
- Criação de ficheiros para a aplicação BEAMnrc .
- Estes ficheiros BEAMnrc simulam as diferentes posições (ângulos) da cabeça do acelerador linear assim como as posições das folhas do "collimator" utilizado no tratamento.

Etapa 2

- Execução de um elevado número de simulações que podem correr em paralelo (individualmente pequenas).
- São necessários aproximadamente 3 minutos num processador 2.27 Nehalem.
- As tarefas são transmitidas como arrays ao OGS para prevenir problemas de calendarização.
- Cada tarefa transfere os dados de input do volume de dados partilhado para o disco local temporário no início por forma a executar localmente.
- Retorna os dados de output para o volume de dados partilhado após a execução.

Etapa 3

- Execução no Master Node.
- Adquire os dados de output da simulação do acelerador e cria novos ficheiros de input para a simulação de paciente que serão utilizados pela aplicação DOSXYZnrc.
- São transferidos mais de 10GB de dados.

Etapa 4

- Novamente cada tarefa copia os ficheiros de input para o disco local e depois armazena os ficheiros de output no volume partilhado.

- As tarefas de maior duração são executadas, levando aproximadamente mais de 40 minutos cada uma no processador.

Etapa 5

- Todos os ficheiros de output são pós-processados no Master Node.
- São unidos num ficheiro de dosagem único (na ordem das dezenas de MB), terminando o processo de verificação.
- O sistema de controlo externo deteta o fim da verificação, realizando o download do output final e undeploy do cluster.

Backend para Cloud

- A solução implementada para a transferência de dados calculados em backend para a 'Cloud' consiste na criação de um cluster virtual para cada pedido.

- Isto garante que cada simulação de tratamento é processada independemente solucionando problemas anteriores detetados nos clusters locais, tal como a interferência entre dois pedidos em simultâneo.

Exemplo

- Utilizando o sistema de Tomografia por computador que é distribuído pelo Hospital da Universidade de Würzburg, foi definida a execução da verificação de tratamentos.
- Para experiências normais, o número de partículas a simular é reduzido, o que significa que uma verificação completa pode ser executada em poucas horas.

- **LINK:**

  http://www.daten.strahlentherapie.uni-wuerzburg.de/quasimodo.html

[3.0] Experiências/Testes

- Foram executadas 3 experiências para verificar se a arquitetura implementada funciona corretamente:

- Distribuição "Multi-site";
- Caso com um objetivo específico definido ("deadline");
- Sistema de Tolerância a Falhas;

- Antes da sua execução foi realizada uma seleção para os parâmetros do Motor de Elasticidade.

[3.1] Elasticity Engine

- Os parâmetros do elasticity engine dependem da infraestrutura a utilizar assim como da aplicação. No exemplo que se segue, o algoritmo é aplicado independentemente nas fases 2 e 4 do eIMRT e para cada fase:
- "Key Application Indicator (KAI)" é número total de partículas iniciais a simular (referenciadas como histórias);
- DTL é o limite de tempo para cada fase;
- Monitorização fornece o rácio correspondente para cada nó em histórias por segundo (hist/s);

Parâmetros

- ELASTICITY_PERIOD
- DEPLOYMENT_PERIOD
- UNDEPLOYMENT_PERIOD
- LOW_PERFORMANCE_INTERVAL
- MAX_CORES
- MARGIN

[3.2] Distribuição Multi-site

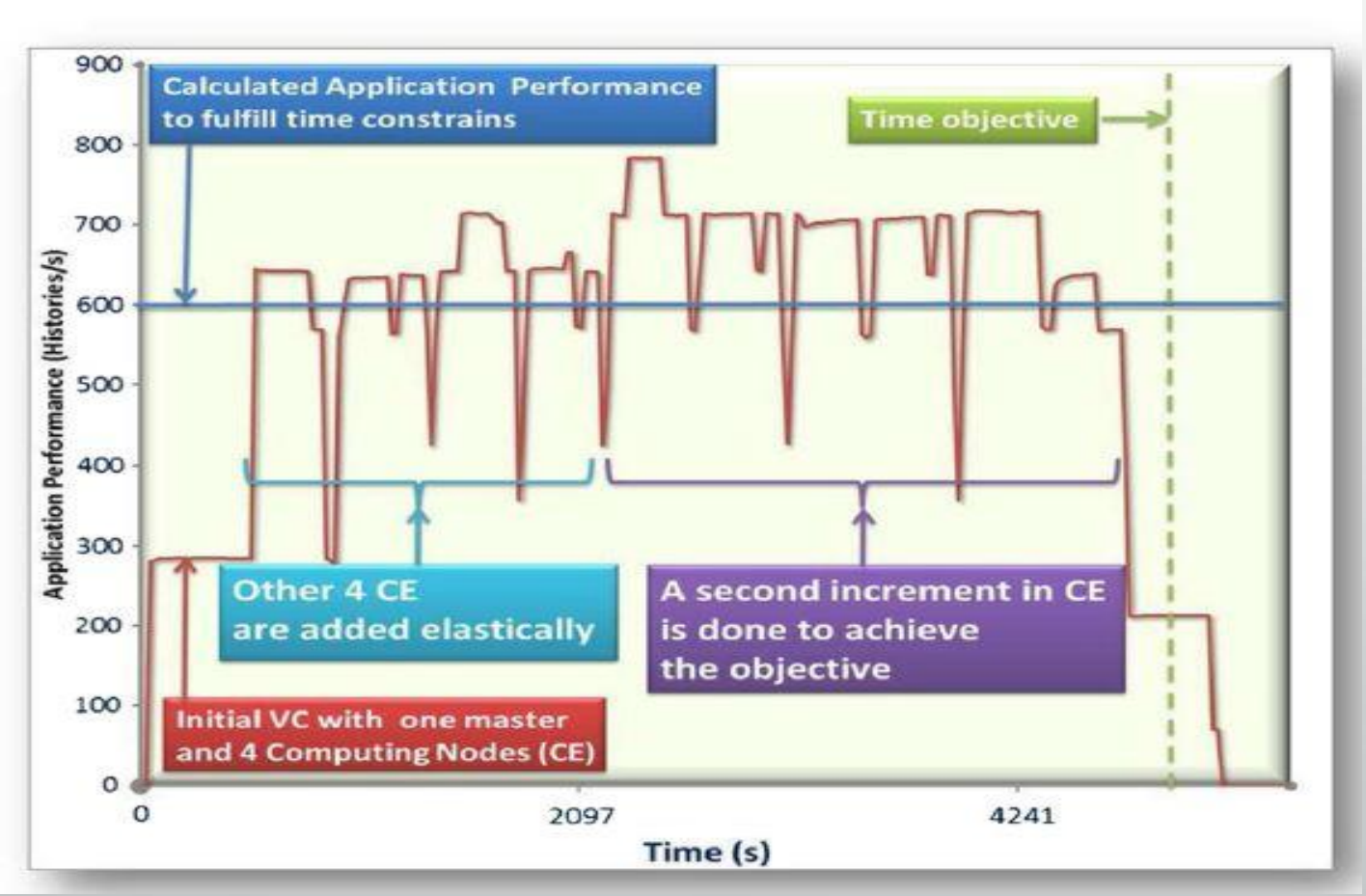
- Foi distribuído um cluster virtual em dois 'sites' BonFIRE:

- INRIA (master node e 7 CEs)
- HLRS(apenas 8 CEs)

- O HLRS apesar de ser mais lento na distribuição das máquinas é mais rápido na execução da aplicação (processador e nós físicos muito superiores).

[3.3] Objetivo definido ("Deadline")

- O objetivo era terminar a fase 2 em 5.000 segundos para 3.000.000 de histórias. Ou seja o rácio desejado seria de 600 histórias por segundo (h/s).

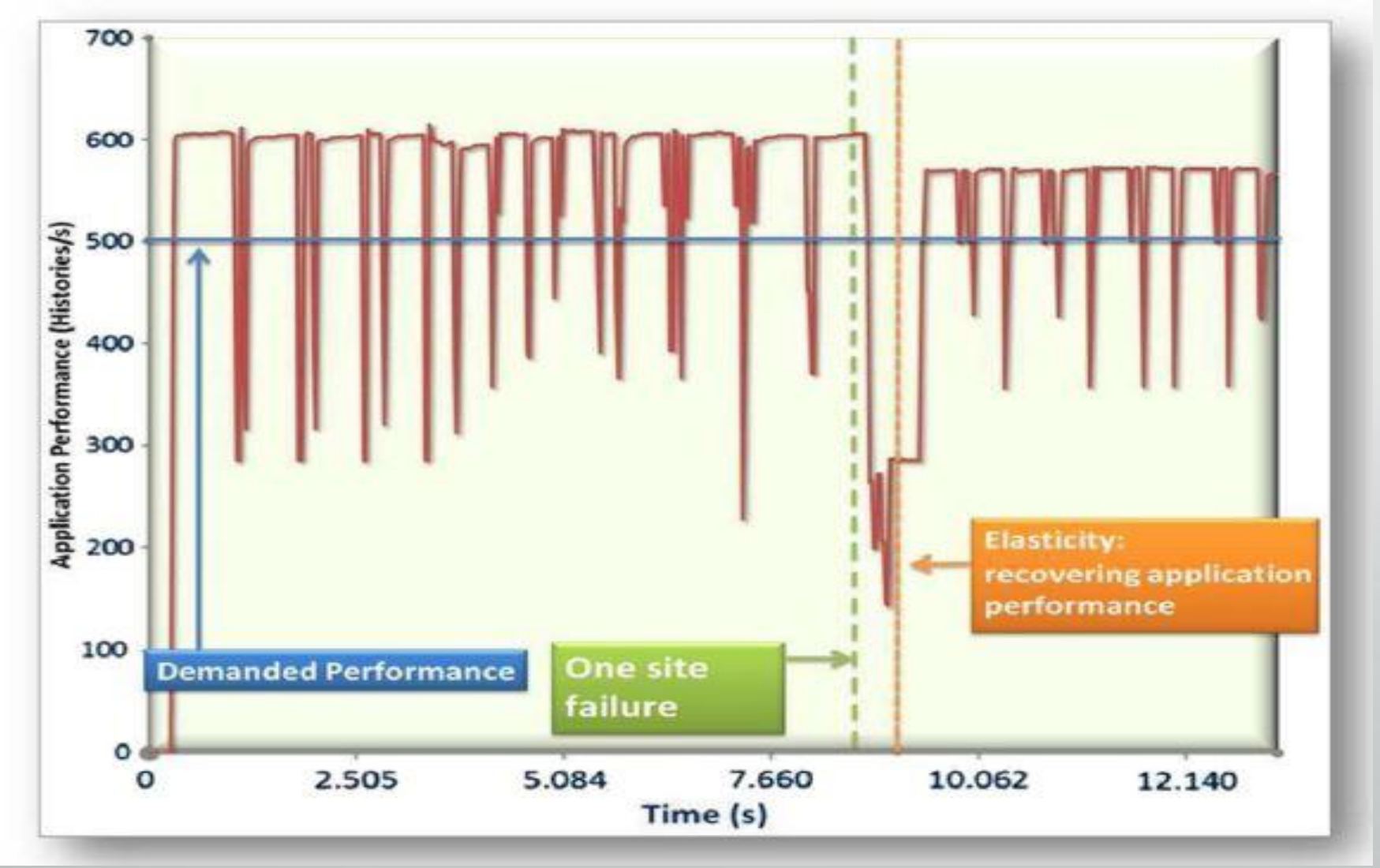


[3.4] Sistema de Tolerância a Falhas

- O sistema completo de tolerância a falhas foi implementado entre as instâncias INRIA e HLRS com 4 nós de trabalho em cada uma. O master node foi implementado no HLRS enquanto que o shadow node corria na INRIA.

- O diagrama que se segue descreve um exemplo de uma falha induzida no sistema por forma a testar a forma como este responde a possíveis falhas.

[3.4] Sistema de Tolerância a Falhas



[3.5] Resultados

- A arquitetura de virtual cluster proposto preencheu os requisitos estabelecidos:

- Pode ser implementada em ambientes single cloud ou multi-cloud utilizando diferentes configurações;

- Pode utilizar o indicador de performance da aplicação para ativar elasticidade horizontal

- Pode ser implementada numa configuração multi-site com tolerância a falhas que consegue recuperar de uma falha.

- Apesar do teste "deadline" não ter sido alcançado antes do fim do DTL, o elasticity engine funcionou como esperado, adaptando o tamanho do cluster de forma dinâmica por forma a atingir a performance desejada para a aplicação.

[4.0] Outras Implementações

- Tordsson
- Montero
- Omer
- Yang
- Niehorster

[5.0] Conclusões

- Esta arquitetura pode ser implementada de vários modos diferentes utilizando plataformas cloud IaaS single site ou multi site:

- single site;

- distributed sites;

- single sites e distributed sites tolerantes a falhas;

- A arquitetura foi complementada com um Elasticity Engine que utiliza a performance da aplicação como um trigger para decidir entre aumentar ou diminuir o tamanho do cluster

[5.0] Conclusões

- Foi demonstrado que a performance da aplicação pode ser utilizada para tomar decisões de elasticidade, resultando no aumento do VC por forma a atingir o deadline, apesar ser necessária alguma metodologia para calibrar os vários parâmetros.
- As caraterísticas do sistema de tolerância a falhas foram testadas na simulação de falha de um dos sites da Cloud, recuperando a performance da aplicação sem uma falha geral da simulação.
- A contra partida do Elasticity Engine é depender dos valores dos parâmetros do fornecedor de Cloud e da sua infraestrutura. Como tal deveria ser desenvolvido um algoritmo de Elasticity Engine melhor, com menos parâmetros e independente do fornecedor de Cloud.