Grades Computacionais: Conceitos Fundamentais e Casos Concretos

Fabio Kon e Alfredo Goldman
Departamento de Ciência da Computação
IME – USP

Jornada de Atualização em Informática (JAI)

Congresso da SBC - Belém do Pará
15 e 16 de julho de 2008

Roteiro do minicurso

1) Histórico e Motivação
2) Conceitos Básicos
    * Tipos de Grades
    * Principais Serviços
3) Modelos de Programação
4) Middleware de grades
5) Grades em funcionamento
6) Pesquisa Atual em Grades

Pré-história das grades

- A necessidade de alto poder de processamento existe há várias décadas.
- Na verdade, nasceu com a Computação.
- Na última década, tornou-se obrigatório para várias ciências.
- *All Science is Computer Science*
  - The New York Times 2001

Necessidades das ciências

- Problemas computacionalmente pesados são muito presentes em
  - Biologia
  - Medicina
  - Física
  - Química
  - Engenharia
- mas também são presentes em
  - Ciências Humanas
  - Economia
  - Arte (música, gráfica)

Evolução das ciências

- Criou-se um círculo virtuoso que
  - expande e melhora a qualidade do conhecimento humano e
  - testa os limites do que é possível realizar computacionalmente.
- Lei de Moore implica que
  - O computador pessoal de hoje é o supercomputador da década passada.

Máquinas paralelas

- Década de 1980: computação de alto desempenho era feita com caríssimas máquinas paralelas
(às vezes chamadas de supercomputadores).
- Eram produzidas em quantidades pequenas (centenas ou milhares).
- Tinham custo atíssimo.
- Acesso muito difícil.

Aglomerados (*clusters*)

- Década de 1990: computadores pessoais popularizaram-se.
- Lei de Moore continuou agindo e levou milhões de computadores potentes a universidades, empresas e à casa das pessoas.
- *Clusters* com hardware de baixo custo.
  - p.ex. Beowulf baseado em Linux.

O nascimento das grades

- Meados da década de 1990:
  - Internet popularizou-se.
- Universidades e institutos de pesquisa com
  - conexões locais estáveis de vários Mbps
  - milhares de computadores
  - dezenas de *clusters*
  - conexões de longa distância estáveis.
- Necessidade de poder computacional sempre crescendo.

A grande idéia

- Por que não interconectar aglomerados de diferentes instituições através da Internet?

- Dificuldades:
  - rede (Internet acadêmica)
  - software
  - protocolos
  - middleware
  - gerenciamento / administração
  - ferramentas
  - segurança
  - heterogeneidade

Grade Computacional

- Nossa definição para Grade (cada um tem a sua):
  - Coleção de aglomerados geograficamente distantes e interconectados através de uma rede de forma a permitir o compartilhamento de recursos computacionais e de dados.
  - Esses aglomerados podem ser dedicados ou não e podem possuir desde 1 até centenas de computadores.

União de comunidades

- Meados da década de 1990:
  Computação em Grade tornou-se uma área de pesquisa combinando aspectos de:
  - Sistemas Distribuídos
  - Computação de Alto Desempenho

- Não era um conceito novo, revolucionário.
- Já se falava sobre isso há muitas décadas.

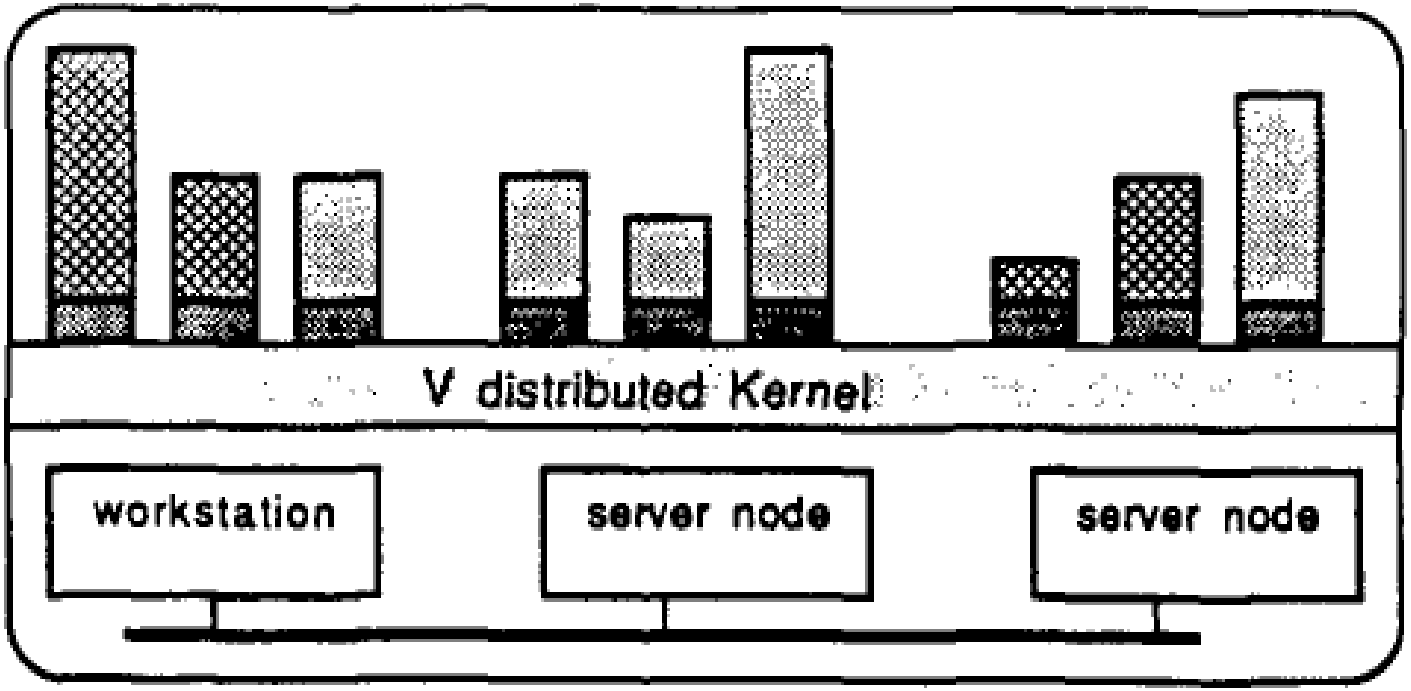
- O termo C*omputational Grid* vem da analogia com *Power Grid*, rede elétrica.

Precursores das grades

Desde a década de 1980, já se falava muito em Sistemas Operacionais Distribuídos:

- Sprite (UCB [Ousterhout et al. 1988])
- NOW (UCB [Anderson et al. 1995])
- Amoeba (Holanda [Tanenbaum et al. 1990])
- Spring (Sun Microsystems, 1993)
- The V System (Stanford [Cheriton 1988])

The V System (1988)



Legend: Run-time library

Command Program

Service Program

FIGURE 1. The V Distributed Operating System

Já na comunidade Paralela

- Algoritmos paralelos [Leighton 1991]
- Máquinas paralelas [Leighton 1991]
  - dedicadas com centenas ou milhares de processadores
- Algoritmos de escalonamento [Bazewicz et al. 2000]
- Memória compartilhada distribuída [Nitzberg and Lo 1991]

Principais diferenças

Computação de alto desempenho tradicional	Grades computacionais
• Recursos dedicados	• Recursos compartilhados
• Configuração estática	• Configuração dinâmica
• Gerenciamento central	• Gerenciamento distribuído
• Ambiente controlado	• Ambiente não controlado
• Máquinas homogêneas	• Máquinas heterogêneas

Principais diferenças

Computação de alto desempenho tradicional	Grades computacionais
• Dezenas ou centenas de máquinas • Máquinas localizadas no mesmo espaço físico • Falhas pouco freqüentes	• Dezenas de milhares de máquinas • Máquinas geograficamente distribuídas • Falhas constantes

Principais Desafios

- Segurança
  - Para os donos dos recursos
  - Para os usuários da grade
- Tolerância a Falhas
  - Para os componentes da grade
  - Para as aplicações em execução na grade
- Escalonamento
  - Determinar onde uma aplicação deve executar
- Gerenciamento

Tipos de grades computacionais

- Grade de Computação
- Grade de Computação Oportunista
- Grade de Dados
- Grade de Serviços
  - “Colaboratórios”
- Grades Móveis

Grades Computacionais

- Foco principal: solução de problemas computacionalmente pesados
- Gerenciamento de recursos
  - Busca de recursos computacionais
  - Entrada e saída de recursos computacionais
- Gerenciamento de Aplicações
  - Escalonamento e ciclo de vida

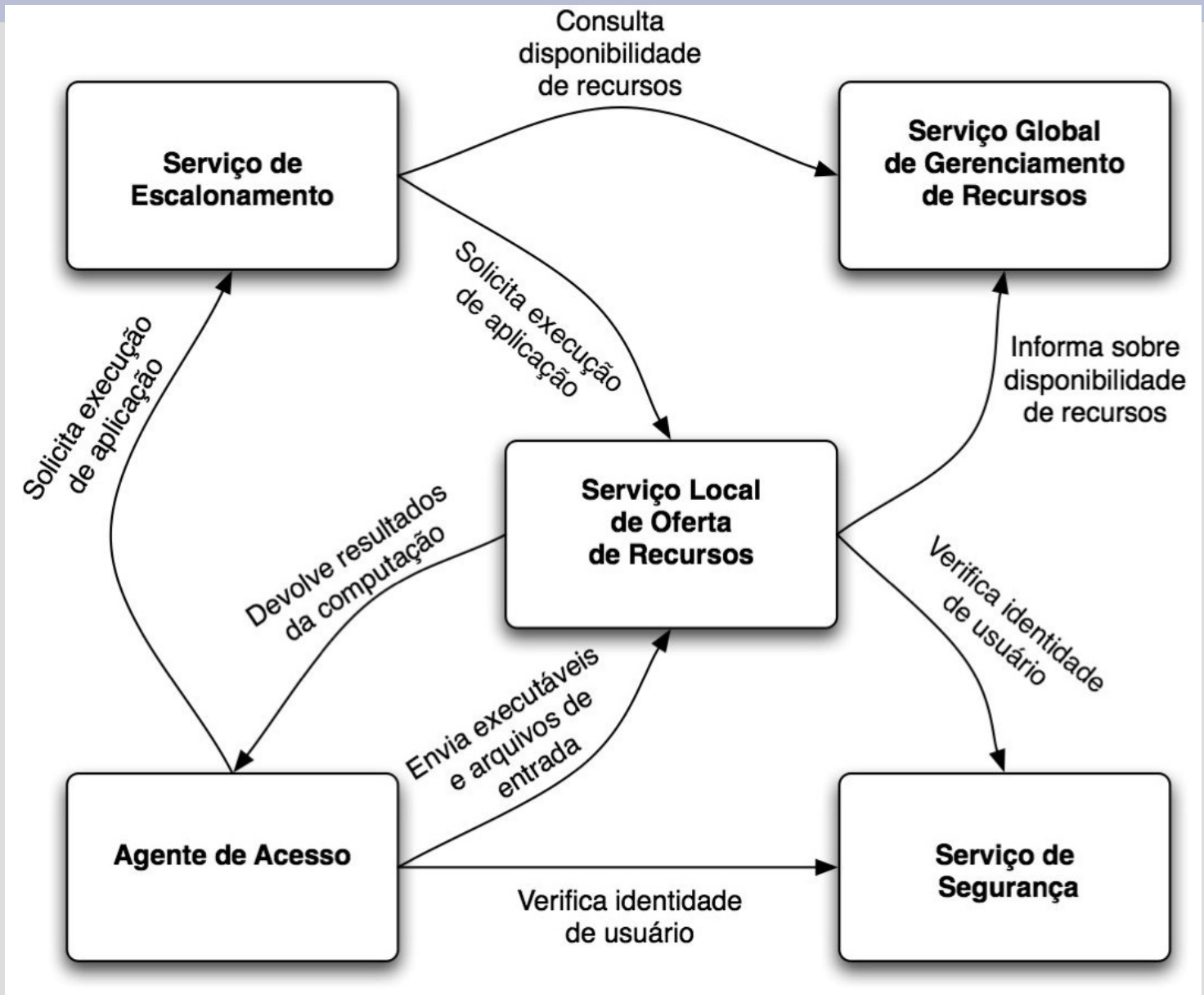
Grades de Dados

- Foco principal: gerenciamento e distribuição de dados.
- Física de Altas Energia
  - Dados gerados em um acelerador de partículas são utilizados por pesquisadores do mundo todo.
  - Precisam gerenciar Petabytes de dados
    - Armazenamento e transmissão dos dados
    - Tolerância a falhas (Replicação)
    - Serviços de busca e diretórios de dados

Grades de Serviços

- Oferecem serviços viabilizados pela integração de recursos na grade:
  - Ambiente de trabalho colaborativo
  - plataforma de aprendizado à distância
  - Colaboratórios para experiências científicas
- A longo prazo, a tendência é que todos esses tipos diferentes de grades serão reunidos em uma única plataforma.

Serviços de uma grade



Agente de acesso

- Ponto de acesso para os usuários interagirem com a grade.
- É executado na máquina do usuário da grade.
- Pode também ser um sistema Web acessado a partir de um navegador.
- Deve permitir que o usuário determine a aplicação a ser executada, os parâmetros da execução e os arquivos de entrada.

Serviço local de oferta de recursos

- Executado nas máquinas que exportam seus recursos à Grade.
- Gerencia uso local de recursos.
- Responsável por executar aplicações nos nós da Grade. Precisa então
  - obter o arquivo executável
  - iniciar execução
  - coletar e apresentar eventuais erros
  - devolver os resultados da execução

Serviço global de gerenciamento de recursos

- Monitora o estado dos recursos da grade.
- Responde a solicitações por utilizações dos recursos.
- Para tanto, pode utilizar outros serviços, tais como.
  - Serviço de informações sobre recursos
    - coleta informações sobre recursos tais como capacidade total e capacidade corrente de processador, RAM, espaço em disco, largura de banda, etc.
  - Serviço de escalonamento

Serviço de escalonamento

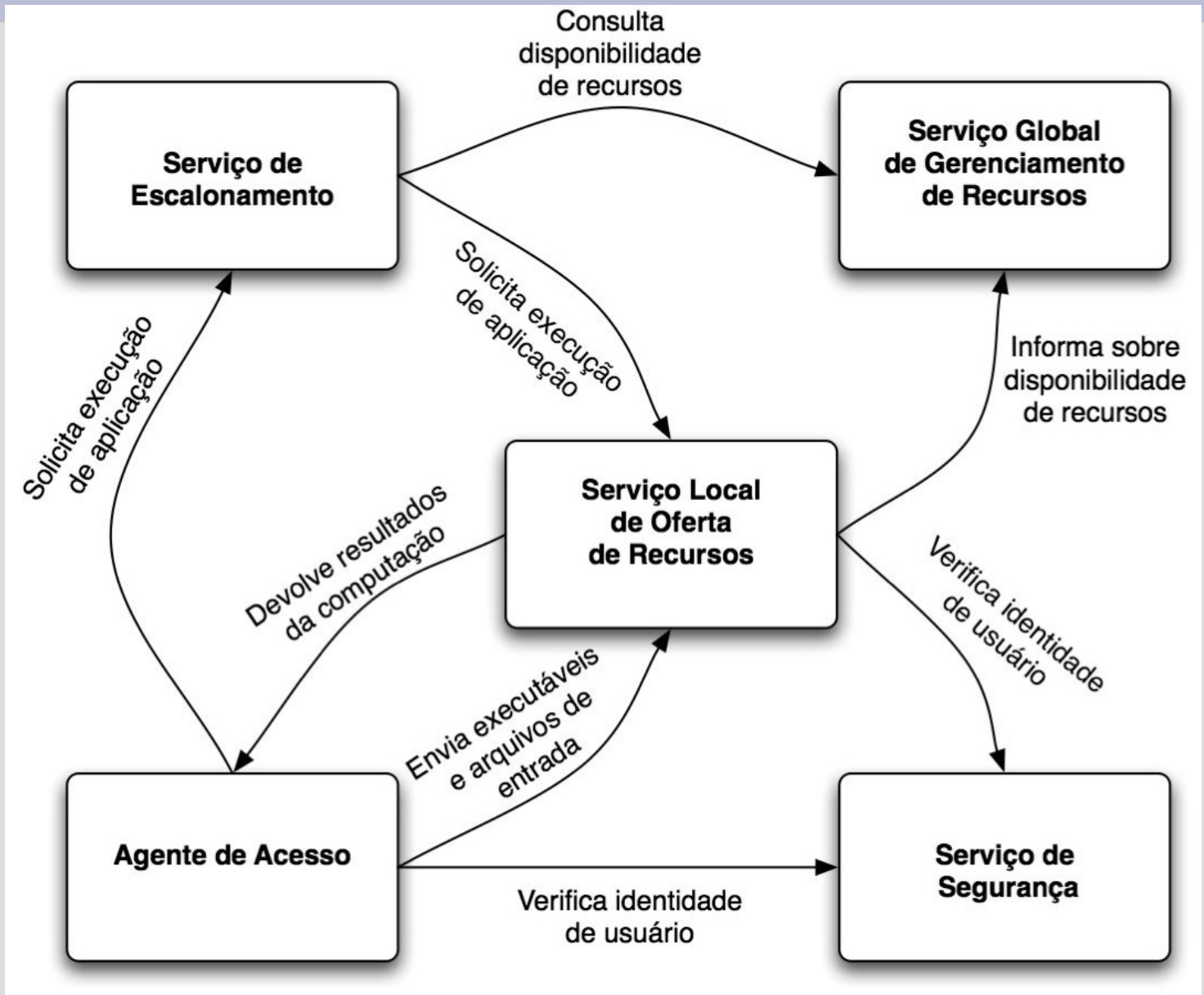
- Determina onde e quando cada aplicação será executada.

  - Recebe solicitações de execução.
  - Obtém informações sobre disponibilidade de recursos.
  - Ordena a execução das aplicações

Serviço de segurança

- Protege os recursos compartilhados, evitando ataques aos nós da grade.
- Autentica os usuários de forma a saber quem é responsável por cada aplicação e por cada requisição de execução.
- Fornece canais seguros de comunicação e de armazenamento de dados e aplicações garantindo confiabilidade e integridade.

Serviços de uma grade



Gerenciamento de recursos

- A principal tarefa de um SO é gerenciar e proteger os recursos de uma máquina.
- A principal tarefa de um middleware de grades é gerenciar e proteger os recursos de uma grade
  - i.e., um conjunto, potencialmente muito grande de máquinas.
- O primeiro ponto é conhecer quais os recursos disponíveis em cada instante.

Recursos mais comuns

- No contexto de alto desempenho, os recursos mais explorados são:
  - processador
  - memória principal
- Mas em alguns casos, também são fundamentais:
  - conexões de rede
  - armazenamento persistente (ex. discos)

Outros recursos

- Placas de processamento de áudio
- Compressores de vídeo
- Monitores de altíssima qualidade
- Telescópios eletrônicos
- Radiotelescópios
- Servidor de armazenamento com petabytes
- Sensores de temperatura, pressão, luminosidade
- Et cetera.

Informações estáticas de um recurso

- Processador
  - tipo: PowerPC G5, Pentium 4.
  - velocidade: 3,8 GHz.
- Sistema operacional
  - tipo: Linux, Windows, MacOS, Solaris, Irix
  - versão: Debian/kernel 2.6.20, NT, 10.5.3
- Memória principal e secundária
  - RAM total de 8GB
  - disco de 250GB

Informações dinâmicas de um recurso

- Processador
  - utilização= 34%
- Memória principal e secundária
  - RAM disponível = 3,4 GB
  - memória virtual disponível = 8,7 GB
  - disco disponível = 232GB

Heterogeneidade

- Mas as grades são heterogêneas.
- Onde uma aplicação será executada mais rapidamente?

  1) Nó com PowerPC G5 de 2,0 GHz e 90% do processador livre e 1GB de RAM livre ou
  2) Nó com Pentium 4 de 3,8 GHz e 80% do processador livre e 2GB de RAM livre.

Comparações de desempenho em arquiteturas diferentes

- Pode-se usar ferramentas tais como bogomips ou outros *benchmarks*.
- Mas ainda é um problema não totalmente resolvido.
- O desempenho do processador também varia de acordo com a aplicação.
  - Há alguns anos processadores AMD eram mais rápidos que os da Intel para processamento de texto e mais lentos para processamento numérico.

Monitoramento

- Serviço local de oferta de recursos
  1) monitora os recursos locais (ex. a cada 5 segundos)
  2) envia dados sobre a máquina local para o serviço global (ex. a cada 5 minutos ou mudanças significativas).
- O serviço global agrega as informações em bancos de dados, potencialmente com informações sobre milhares de nós.
- Serviço global pode ser centralizado, ou estruturado de forma hierárquica, ou par-a-par, etc.

Federação de aglomerados

- Um grande desafio é estruturar o mecanismo de monitoramento de forma que ele não se transforme em um gargalo quando o sistema cresce.
- Grades modernas são organizadas em federações de aglomerados estruturadas como árvores hierárquicas ou estruturas P2P.
- Cada aglomerado possui seus gerenciadores locais.
- A federação conta com protocolos inter-aglomerados para monitoramento e escalonamento.

Grades Computacionais:

Conceitos Fundamentais e Casos Concretos

Algoritmos de escalonamento e escalonadores

Fabio Kon, Alfredo Goldman

Universidade de São Paulo
{kon,gold}@ime.usp.br

SBC - JAI - Belém - 15 e 16/7/08

Roteiro

Definição de escalonamento

Nomenclatura e arquitetura

Algoritmos de Escalonamento

Exemplos de escalonadores
- SLURM
- OAR

Escalonamento

Em inglês *scheduling*, corresponde ao estudo de como alocar tarefas a recursos.

- Tanto as tarefas como os recursos podem ser de tipos diferentes;
- a alocação otimiza uma função de custo previamente definida;
- mas, geralmente, esses problemas são difíceis.

Entre os vários tipos de tarefas podemos ter processos, *threads* ou mesmo espaço em disco.

Recursos podem ser computacionais, de armazenamento, de rede, etc,

Um problema difícil 1/2

Um exemplo simples da dificuldade do problema do escalonamento é o problema da partição, que é NP-Completo. Dados :

- n e n inteiros $I = \{i_1, \ldots, i_n\}$

Queremos dividir o conjunto I em dois conjuntos I_1 e I_2, $I_1 \bigcup I_2 = I$ e $I_1 \cap I_2 = \emptyset$ tais que :

$$\sum_{i \in I_1} i = \sum_{i \in I_2} i$$

Dos problemas difíceis este é um dos mais fáceis (pseudo-polinomial).

Um problema difícil 2/2

No nosso contexto, os inteiros i_j podem ser vistos como o tamanho de tarefas seqüenciais a serem alocadas para duas máquinas idênticas.

É fácil imaginar que com tarefas paralelas e ambientes dinâmicos, o problema fica bem mais difícil.

É interessante notar que existem problemas « ainda » mais difíceis como a 3-partição :
Dados os inteiros positivos $A = (a_1, \ldots, a_{3t})$ e B, existe uma partição $(T_1, \ldots, T_t)$ de A tal que $|Tj| = 3$ e $\sum_{a_i \in T_j} a_i = B$ para $j = 1, .., t$?

O que fazer com problemas difíceis ?

Encontrar a solução exata de um problema difícil pode levar muito tempo.
Quais as possíveis abordagens :

- Aproximações : ao invés de buscar a solução ótima procura-se encontrar uma solução que esteja a um fator dela.
  (ex. : no máximo 2 vezes o tempo da solução ótima)
- Heurísticas : algoritmos que não fornecem nenhuma garantia em relação à solução, mas que podem funcionar em diversas situações.

Em grades, onde o ambiente é geralmente heterogêneo e dinâmico, o melhor é procurar aproximações e heurísticas.

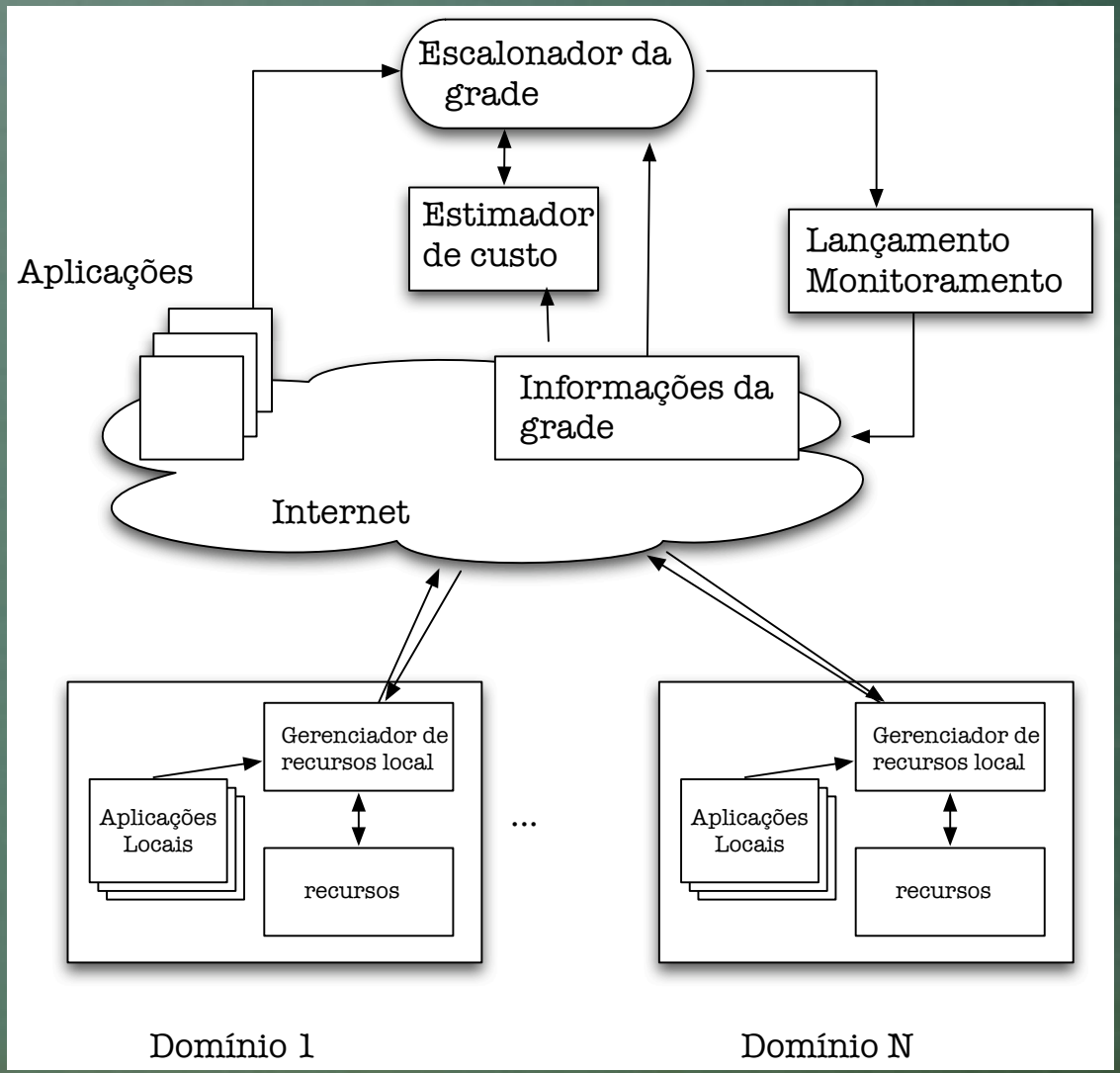
Nomenclatura

Definições geralmente usadas em escalonamento para grades :

- **tarefa** : unidade indivisível a ser escalonada
- **aplicação** : conjunto de tarefas e aplicações
- **recurso** : algo capaz de realizar alguma operação
- **sítio** : unidade autônoma da grade, composta por recursos

escalonamento de tarefas é um mapeamento, no tempo, das tarefas para recursos.
Conceito importante : escalonamento *válido* (atentar para as capacidades)

Arquitetura Genérica



Algoritmos

Enquadram-se em dois possíveis contextos :

Estático : conhecimento total das aplicações a serem submetidas antes do seu início.

Dinâmico : as aplicações podem aparecer a qualquer momento; as próprias tarefas das aplicações podem não ser conhecidas.

Algoritmos estáticos existem mas são pouco apropriados para grades :

- recursos podem mudar;
- o tempo previsto de uma tarefa pode não corresponder ao esperado;
- para certas aplicações, não é possível prever o seu tempo de execução;
- o modelo estático pressupõe um conhecimento total a priori.

Algoritmos dinâmicos

- Conforme estimativas, as tarefas são alocadas a recursos.
- Podem ser alocadas no momento de sua criação.
- Para uma adaptação dinâmica usa-se Balanceamento de carga :
  - tentativa de distribuir o trabalho entre os recursos;
  - mais comum : roubo de trabalho (quem não tem, busca)

1. Adaptabilidade ao ambiente.
2. Procura-se maximizar o uso dos recursos (ao invés do tempo de término, que favorece um único usuário).

Escalonadores de Grades

Duas abordagens principais :

- Escalonadores completos (global + local).
- Meta-escalonadores que « conversam » com escalonadores locais.

Veremos dois :

- SLURM (Lawrence Livermore National Laboratory, início 2002)
- OAR (Mescal Project - INRIA/UJF, início 2002)

SLURM

Simple Linux Utility for Resource Management

- aglomerados heterogêneos com milhares de nós Linux
- provê tolerância a falhas
- oferece
  1. controle de estado das máquinas
  2. gerenciamento de partições
  3. gerenciamento de aplicações
  4. escalonador
  5. módulos para cópia de dados

`slurmd` - executado em cada nó do sistema

`slurmctld` - executado no nó de gerenciamento

SLURM (continuação)

Funções principais :

- alocar recursos, de forma exclusiva, ou não, por períodos determinados
- fornecer um arcabouço para lançar, executar e monitorar as aplicações nas máquinas
- gerenciar uma fila de trabalhos pendentes no caso de pedidos de uso conflitantes

Escalonador simples do tipo FIFO com prioridades
Diferencial : Alto desempenho (usado no BlueGene/L), aceita plug-ins.

OAR

Escalonador para aglomerados e grades

- Permite a reserva de máquinas de uma grade por usuários específicos.
  Para ele, os recursos são as máquinas.
- Possui comandos para :
  1. submissão
  2. cancelamento
  3. consulta
- Há controle de admissão e a aplicação é enfileirada.
  É executada quando os recursos tornam-se disponíveis.

OAR (continuação)

- Escalonamento modular, configurável pelo administrador :
  - filas de submissão diferentes (prioridades)
  - casamento de recursos
  - *backfilling* (fura fila se não atrapalha)
- Permite reservas
- Existe uma versão para ambientes oportunistas

Diferencial : Simplicidade e modularidade. É usado no Grid 5000.

Segurança

- Com a popularização da Internet e a conexão dos mais variados sistema a ela, a Segurança se tornou um problema fundamental.
- Grandes empresas investem quantias vultosas para proteger seus sistemas de ataques externos (e internos).
- Por sua natureza, grades computacionais podem escancarar a porta de entrada aos sistemas. Portanto, segurança é fundamental.

Múltiplos domínios

- Grades podem cobrir diferentes domínios administrativos.

- Isso dificulta o gerenciamento de
  - usuários
  - permissões
  - controle de acesso
  - contabilização de uso de recursos
  - auditoria
  - privacidade

Objetivos do Serviço de Segurança

- Autenticação
  - processo de estabelecer a validade de uma identidade reivindicada;
  - deve-se autenticar administradores, provedores de recursos, provedores de aplicações, usuários da grade.
- Confidencialidade
  - impede que os dados sejam lidos ou copiados por usuários que não têm esse direito.
  - ex.: grades entre empresas ou bancos.

Objetivos do Serviço de Segurança

- Integridade de Dados
  - proteção da informação, evitando que ela seja removida ou alterada sem a autorização de seu dono.

- Disponibilidade
  - proteção dos serviços, evitando que eles sejam degradados a ponto de não poderem ser mais usados.
  - proteção a ataques de negação de serviço.
  - ex.: usuário deve ser capaz de acessar o resultado de suas computações e de executar novas aplicações na grade.

Objetivos do Serviço de Segurança

- Controle de Acesso
  - permite que apenas usuários conhecidos, e com os respectivos direitos de acesso, disponham dos recursos da grade;
  - o gerenciamento de direitos de acesso de centenas de usuários em milhares de recursos deve ser realizado de forma viável.
  - mecanismos utilizados
    - listas de controle de acesso
    - certificados de direitos
    - capacidades (*capabilities*)
    - controle de acesso baseado em papéis (RBAC)

Objetivos do Serviço de Segurança

- Auditoria
  - registro de todas ações relevantes executadas no sistema indicando quem fez o que e quando
  - sistemas de segurança falham; a auditoria provê formas de detectar quando a falha ocorreu, qual falha ocorreu, etc.
  - agentes inteligentes podem monitorar os registros de forma a detectar intrusos.

- Irretratabilidade (*non-repudiation*)
  - obtenção de provas (ou fortes indícios) das ações de um usuário de forma que ele não as negue.

Mecanismos de segurança

- Canais de comunicação seguros
  - criados entre os nós da grade de forma que dados não sejam roubados ou corrompidos
  - atualmente, o meio mais comum é SSL (*secure socket layer*) implementados por bibliotecas como OpenSSH
- Armazenamento seguro de dados
  - dados armazenados de forma persistente não podem ser perdidos nem acessados por usuários não-autorizados.
  - Criptografia e proteção do SO local ajudam.

Mecanismos de segurança

- *Sandboxing*
  - como um areião onde crianças brincam, isola as aplicações do ambiente exterior; as aplicações podem fazer a farra que quiserem pois o estrago estará limitado;
  - limitam
    - uso de memória
    - uso de disco (quantidade e porção do disco)
    - uso do processador
    - uso da rede (banda e aonde se conectar)
  - A máquina virtual Xen [Barham et al. 2003] tem sido muito usada para isso.

Tolerância a Falhas

- Grades são um ambiente altamente dinâmico. Nós podem entrar e sair a qualquer momento.
- Aplicações pesadas podem demorar horas, dias, ou meses para serem executadas.
- Se um único nó de uma aplicação paralela falha, toda a computação pode ser perdida.

- Portanto, tolerância a falhas é essencial !

Tolerância a falhas

- Além disso, os próprios nós que executam os serviços da grade podem falhar.
- Portanto, é necessário que haja replicação dos serviços essenciais da grade.

- Em ambos os casos, é necessário um bom mecanismo para **detecção de falhas**.
- Este mecanismo deve informar o middleware da grade em caso de falhas para que as aplicações sejam reiniciadas ou os serviços reestabelecidos.

Checkpointing

- Consiste em tirar fotografias do estado de um processo de forma a constituir *pontos de salvaguarda*.

- Permite reiniciar a aplicação de um ponto intermediário de sua execução.

- Em aplicações seqüenciais é trivial.
- Em aplicações paralelas com comunicação entre os nós é mais complicado.

Recuperação por Retrocesso

- Recuperação por retrocesso baseada em *checkpointing:*
  - *Checkpoints* contendo o estado da aplicações são gerados periodicamente.
  - *Checkpoints* são armazenados nas máquinas do aglomerado.
  - Em caso de falha durante a execução, o middleware reinicia a aplicação a partir do último checkpoint gerado.

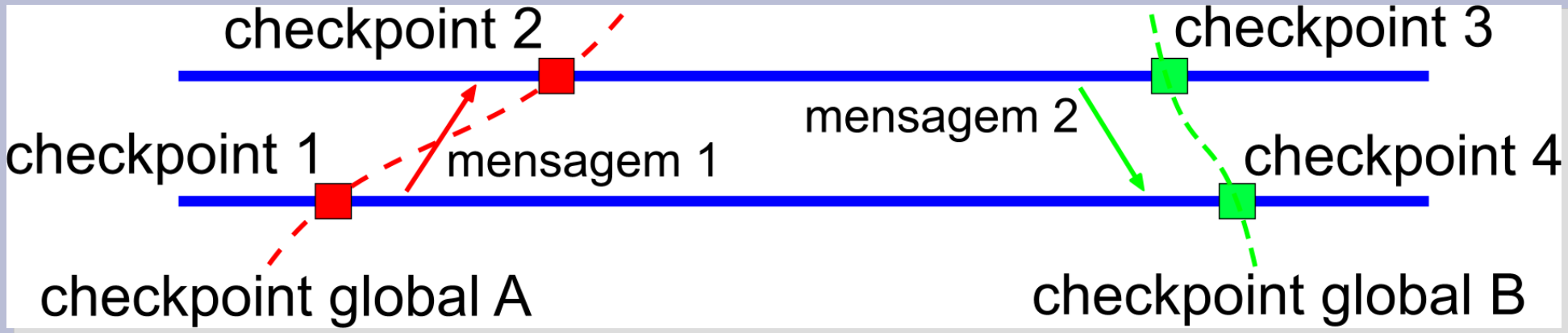
Abordagens para *checkpointing*

- O estado de uma aplicação pode ser obtido de diferentes modos.
- *Checkpointing* no nível do sistema (ex. CryoPID)
  - Dados obtidos diretamente do espaço de memória
    - Transparente à aplicação
    - Requer uso de máquinas homogêneas
- *Checkpointing* no nível da aplicação
  - Aplicação fornece dados a serem salvos
    - *Checkpoints* portáteis
    - Precisa modificar código-fonte da aplicação

Checkpointing de Aplicações Paralelas

- *Checkpoints* locais → *checkpoint* global
- Aplicações paralelas desacopladas
  - Aplicações *bag-of-tasks*
  - Basta gerar um *checkpoint* para cada processo
- Aplicações paralelas acopladas
  - Aplicações BSP e MPI
  - Dependências entre processos
  - É preciso coordenar a criação de *checkpoints* locais

Checkpointing de Aplicações Paralelas



- *Checkpoint* 1 é inconsistente e 2 é consistente
- Protocolo coordenado de *checkpointing*
  - Sincroniza os processos antes de gerar *checkpoint*
  - Deste modo os *checkpoints* gerados são sempre consistentes

Coordenado vs. assíncrono

- *Checkpointing* coordenado é relativamente simples de implementar mas o desempenho é pior.
- *Checkpointing* assíncrono permite que se tire as fotografias sem interromper a execução dos processos.
  - mas é muito mais complexo.
  - ainda não são usados no contexto de grades.

Grades Computacionais:

Conceitos Fundamentais e Casos Concretos

Modelos de Programação

Fabio Kon, Alfredo Goldman

Universidade de São Paulo
{kon,gold}@ime.usp.br

SBC - JAI - Belém - 15 e 16/07/08

Roteiro

- Preliminares
- Paramétricas e saco de tarefas
  - SETI@home
  - Mersenne Prime Search
- MPI
  - Programação MPI
- BSP
  - Programação BSP
- Outros modelos
  - OpenMP
  - KAAPI

Modelos de Programação

Abstração da máquina de forma a simplificar a representação da realidade.

- Modelos mais próximos da realidade são muito complexos.
- Modelos excessivamente simplificados podem estar longe da realidade.
- É necessário um balanço entre a realidade e a simplicidade.

Veremos alguns modelos úteis para representar aplicações.

Modelos de Programação (analogia)

Modelos paralelos : LogP (e variantes), PRAM, ...
Existem cursos específicos, como :

- Topics in Parallel Algorithms using CGM/MPI - Song, Cáceres e Mongelli, SBAC 03.
- Modelos para Computação Paralela - Goldman, ERAD 03.

Analogia com seqüencial

- Mais simples : Modelo de Von Neuman
- Mais complexo : considerar os tempos de acesso à memória, caches de diversos níveis, etc.

Paramétricas e saco de tarefas

- Aplicações paramétricas correspondem a uma classe de aplicações idênticas que são executadas com diferentes parâmetros.
- Aplicações do tipo saco de tarefas (*bag-of-taks*) correspondem a aplicações onde as tarefas a serem executadas são completamente independentes umas das outras.
- A necessidade de comunicação entre tarefas é mínima.

Duas comunicações

1. uma no início, quando são passados os parâmetros iniciais
2. e uma ao término, quando os resultados são transmitidos para serem combinados.

Escalabilidade

Como quase não há comunicação, um grande número de máquinas pode ser usado.

A paralelização é simples (*embarrassingly parallel*).

Algumas áreas de aplicação :

- mineração de dados
- simulações de Monte Carlo
- renderização distribuída
- Buscas com BLAST em bioinformática
- aplicações em física de partículas

SETI@home

Projeto SETI@home (`setiathome.berkeley.edu`)

- computação voluntária
- 387 TeraFLOPS (jan/08)
- Busca de padrões em arquivos
- possui sofisticados mecanismos de controle

BOINC

O sucessor deste projeto é o BOINC (`boinc.berkeley.edu`) para aplicações saco de tarefas genéricas.

Busca de Primos

Projeto de busca de número primos no formato $2^n - 1$

- computação voluntária
- `www.mersenne.org`
- maior primo $2^{32,582,657} - 1$ (9,808,358 dígitos)
- prêmio de 100 mil dólares para quem achar um primo com 10 milhões de dígitos

MPI

Message Passing Interface
Padrão criado pelo MPI Forum (www.mpi-forum.org)

Define interfaces, protocolos e especificações para a comunicação paralela.

Principais objetivos :

Desempenho

Escalabilidade

Portabilidade

Apesar de ser de « baixo nível », ainda é o padrão para computação paralela mais utilizado.

Diferentes implementações

O MPI possui diferentes implementações, sendo as mais conhecidas :

- MPICH - Argonne National Laboratory
- Open-MPI - (mantido por um consórcio) sucessor do mpi-lan

Para atingir alto desempenho, existem implementações específicas para diversas placas de rede

Versões principais :

- MPI 1 (a partir de 1994) - troca de mensagens, ambiente de tamanho fixo
- MPI 2 (a partir de 1997) - criação dinâmica de processos, operações em memória remota

Troca de mensagens

É o principal modelo de comunicação em MPI.

Processos independentes com memórias locais trocam informações através do envio de mensagens.

- envio (`MPI_Send()`)
- recepção (`MPI_Recv()`)
- Existe a necessidade de sincronização
  - Existem sends e receives bloqueantes : a instrução bloqueia o fluxo de execução.
  - Existem também instruções assíncronas : a verificação da transmissão pode ser feita posteriormente (`MPI_Wait()`)

Um programa

```
#include "mpi.h"
#include <stdio.h>
int main(int argc, char **argv) {
   int  rc;
   rc = MPI_Init (&argc, &argv);
   if (rc == MPI_SUCCESS) {
     printf ("MPI iniciou corretamente.\n");
   }
   MPI_Finalize ();
   return 0;
}
```

Execução de um programa

Para compilar executar um programa existem comandos como o `mpicc` e o `mpiexec`.

No caso acima, após a compilação, temos o comando `mpirun [ -np X ] <program>` que executa `X` cópias do programa no ambiente.

O número de mensagens do programa anterior depende de quantas cópias serão lançadas.
Os processos são distribuídos (*round-robin*) entres as máquinas disponíveis no ambente.

Conceitos do MPI

Em MPI, é possível definir « comunicadores » que correspondem a abstrações que permitem a processos relacionados trocar mensagens.
O mais comum é o `MPI_COMM_WORLD` que é comum a todos os processos lançados através de `mpiexec`

As duas instruções seguintes são bem úteis :

- `MPI_Comm_size (comm, *number)` dado um comunicador, fornece sua quantidade de processadores.
- `MPI_Comm_rank (comm, *id)` devolve o número do processo corrente dentro do comunicador

Isso deve ficar claro com o próximo exemplo.

Um programa com « comunicador »

```
int main(int argc, char **argv) {
  int  num_of_processes, rank, rc;
  rc = MPI_Init (&argc, &argv);
  if (rc == MPI_SUCCESS) {
    MPI_Comm_size (MPI_COMM_WORLD, &num_of_processes);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    printf ("Sou o processo %d de %d\n", rank,
                num_of_processes);
  }
  MPI_Finalize ();
  return 0;
}
```

Forma de programação

O modelo de programação em MPI é o SPMD *Single Program Multiple Data.*

Nesse modelo, existe um único programa e, conforme váriaveis locais, o comportamento muda.
Uma das variáveis locais usadas é o id do processo.
Com um simples esquema podemos definir que um « mestre » envia trabalho aos outros nós.

```
if(id == 0}
  // envia trabalho
else
  // recebe e executa
```

Sintaxe para o envio

Vejamos um programa onde um processador envia uma mensagem a outro.
Para isso, usaremos o `MPI_Send` (síncrono)

```
int MPI_Send(void *buf, int count, MPI_Datatype datatype,
             int dest, int tag, MPI_Comm comm )
```

```
Input Parameters
buf:  initial address of send buffer
count:  number of elements in send buffer
datatype:  datatype of each send buffer element
dest:  rank of destination
tag:  message tag
comm:  communicator
```

Sintaxe para a recepção

Sintaxe do MPI_Recv (síncrono)

```
int MPI_Recv(void *buf, int count, MPI_Datatype datatype,
             int source, int tag, MPI_Comm comm,
             MPI_Status *status )
```

Output Parameters
buf: initial address of receive buffer
status: status object
Input Parameters
count: maximum number of elements in receive buffer
datatype: datatype of each receive buffer element
source: rank of source
tag: message tag
comm: communicator

Sincronismo

No caso da recepção síncrona, o comando Bloqueia o fluxo até a recepção completa da mensagem.
Para o envio síncrono, o comando Bloqueia o fluxo em um de três pontos :

- Até o envio completo da mensagem.
  Isto é, a mensagem foi repassada para uma outra camada que não o MPI.
- Até a recepção da mensagem pelo nó destinatário.
  A mensagem é recebida pela máquina em que o processo destino se encontra.
- Até a recepção pelo processo destinatário.
  A mensagem é efetivamente recebida pelo processo.

Não existe na especificação a determinação de onde ocorre o Bloqueio.

Uma troca de mensagens

```
int main(int argc, char **argv) {
  int numtasks, rank, dest, source, rc, count, tag = 1;
  char inmsg, outmsg; MPI_Status Stat;
  MPI_Init (&argc, &argv);
  ...
  if (rank == 0) {
     dest = 1;
     outmsg = 'x';
     printf ("Enviando caractere %c para proc %d\n", outmsg, dest);
     rc = MPI_Send (&outmsg,1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
  }
  else if (rank == 1) {
    source = 0;
    rc = MPI_Recv (&inmsg, 1, MPI_CHAR, source, tag,
                        MPI_COMM_WORLD, &Stat);
    printf ("Recebi o caractere: %c do proc %d\n", inmsg, source);
  }
  MPI_Finalize ();
}
```

Comunicação em grupo

Além da possibilidade de troca de mensagens entre pares de processadores, o MPI oferece possibilidades para a comunicação entre grupos de processadores.

As mais comuns são :

- Difusão - de um para todos os processos do grupo
- Concentração - de todos os processos para um
- Redução - aplica uma operação (ex. : soma) nos dados recebidos de todos os processos
- Troca completa - efetua uma troca completa de mensagens entre todos os processos

Sintaxe da redução

```
int MPI_Reduce (void *sendbuf, void *recvbuf, int count,
                MPI_Datatype datatype, MPI_Op op, int root,
                MPI_Comm comm )
Input Parameters
  sendbuf:  address of send buffer
  count:    number of elements in send buffer
  datatype: data type of elements of send buffer
  op:       reduce operation
  root:     rank of root process
  comm:     communicator
Output Parameter
  recvbuf: address of receive buffer
```

Primitivas de comunicação de grupo

```
int main(int argc, char **argv)  {
   int rank, source = 0, dest = 0, rc;
   int value, totalValue;
   MPI_Init (&argc, &argv);
   MPI_Comm_rank (MPI_COMM_WORLD, &rank);
   if (rank == 0)
      value = 10;
   MPI_Bcast (&value, 1, MPI_INT, source, MPI_COMM_WORLD);
   value *= rank + 1;
   MPI_Reduce (&value, &totalValue, 1, MPI_INT, MPI_SUM,
               dest, MPI_COMM_WORLD);
   if (rank == 0)
      printf ("Valor final calculado: %d\n", totalValue);
   MPI_Finalize();
   return 0;
}
```

MPI 2

Funcionalidades específicas de MPI 2 :

- Um processo pode criar sub-processos dinamicamente (em outras máquinas).
- Operações de acesso a memória remota (*one-side-communications*)
  - primeiro é necessário criar-se « janelas »
  - `MPI_PUT()` e `MPI_GET()`
  - além de outras primitivas de comunicação e sincronização.
- suporte para *threads*.

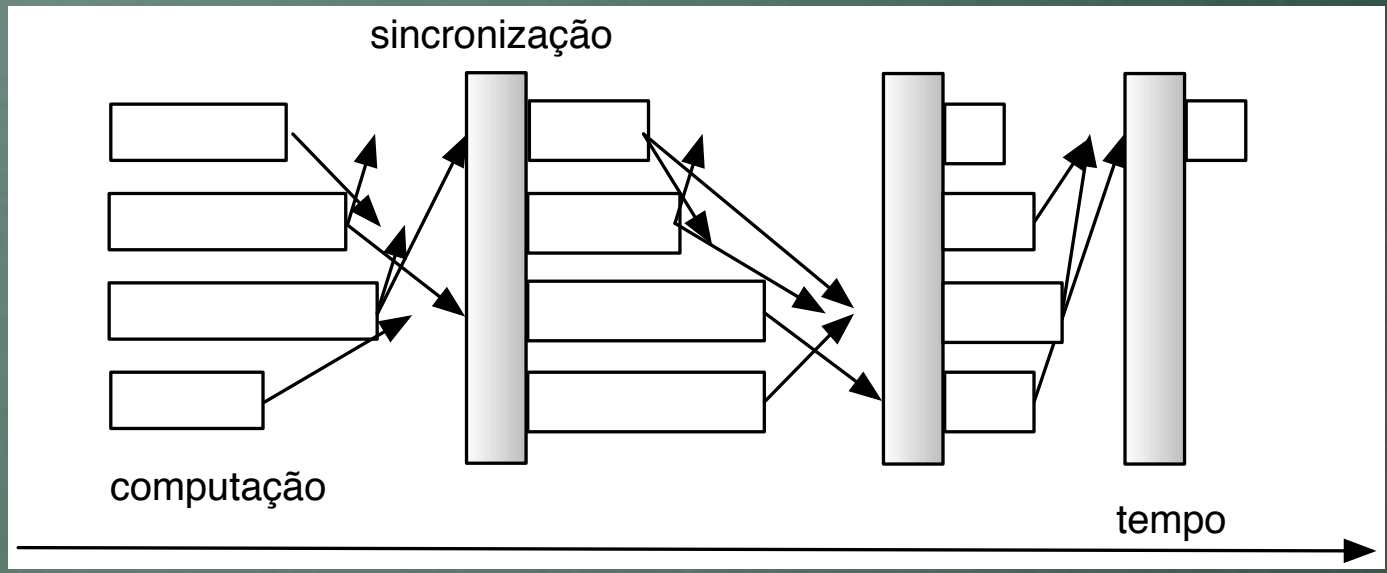
BSP

Bulk Synchronous Parallel Model

Modelo criado por Valiant [1990]

- Objetivo : Portabilidade sem perder o desempenho.
- Idéia : Separação explícita da computação e da comunicação.
- conceitos principais :
  1. super-etapa
  2. sincronização

Esquema de execução BSP



Parâmetros utilizados :

- p --- número de processadores;
- l --- custo de uma sincronização global;
- g --- tempo para transmitir um Byte pela rede. Ou seja, $\frac{1}{g}$ é a Banda passante.

Diferenciais do BSP

Vantagens :

- Conhecendo-se p, l e g é possível estimar o tempo de execução em diferentes plataformas.
- Os pontos de sincronização criam estados globais consistentes.

Desvantagem : com sincronizações globais freqüentes, em alguns casos, os programas podem perder escalabilidade.

Programando com BSP

Para obter um programa eficiente :

- a carga de cálculo deve estar balanceada,
- a comunicação deve ser limitada (quantidade de dados),
- deve-se usar poucas super-etapas.

Modelo semelhante : CGM *Coarse Grained Multicomputers*

Para a programação BSP pesquisadores (principalmente da Oxford University) criaram a BSPlib em 1997.

Formas de comunicação

Duas formas :

- acesso a memória remota -- *Distributed Remote Memory Addressing* (DRMA)
- troca de mensagens -- *Bulk Synchronous Message Passing* (BSMP)
  1. as mensagens são enviadas para a fila local do processo remoto;
  2. cada processo pode acessar sua fila local.

Em DRMA para acessos à memória remota é necessário registrar as regiões a serem acessadas

`bsp_push_reg()`

BSPlib

Comandos da BSPlib :

`Bsp_Begin (nprocs)` --- inicia um programa BSP com *nprocs* processos

`Bsp_end()` --- final da parte paralela

`Bsp_pid()` --- fornece o id do processo

`Bsp_nprocs()` --- fornece o número total de processos

`Bsp_sync()` --- delimitador de super-passos

Leitura e escrita em DRMA :

`Bsp_put()` recebe id do destino, *offset* local, endereço no destino e *offset* remoto

`Bsp_Get()` é semelhante

Produto escalar 1/2

```
#include "BspLib.hpp"
int main (int argc, char **argv) {
  int pid, nprocs = 4;
  double produto,
         *listaProdutos;  // vetor local a cada processo
  listaProdutos = (double *)
                      malloc (nprocs * sizeof (double));
  bsp_begin (nprocs);
  // registra memória a ser compartilha
  bsp_push_reg (&listaProdutos, nprocs * sizeof(double));
  pid = bsp_pid ();
```

Produto escalar 2/2

```
  // cálculo do produto com os pedaços de vetores locais
  prod = calculaProdutoLocal (pid, nprocs);
  // envia os resultados a todos os outros processos
  for (int proc = 0; proc < bsp_nprocs( ); proc++)
    // coloca em proc um valor (double) em
    // listaProdutos usando pid como offset
    bsp_put (proc, &produto, &listaProdutos,
                   pid, sizeof(double));
  bsp_sync ( ); // final do super-passo
  calculaProdutoEscalarFinal (listaProdutos);
  bsp_end( );
}
```

Troca de mensagens

Principais funções :

- `bsp_send(int pid,void *tag,void *payload,int nbytes)` -- envia uma mensagem de tamanho `nbytes` contida em `payload` e com o identificador `tag` para a fila de mensagens do processo `pid`.
- `bsp_set_tagsize(int *nbytes)` -- define o tamanho da `tag` como `nbytes`.
- `bsp_get_tag(int *status, void *tag)` -- se houver mensagens na fila, devolve o tamanho da próxima mensagem em `status` e o conteúdo da *tag* da mensagem em `tag`.

Troca de mensagens (continuação)

- `bsp_move(void *payload, int nbytes)` -- copia o conteúdo da primeira mensagem da fila em `payload`. A variável `nbytes` representa o tamanho do buffer apontado por `payload`.
- `bsp_qsize(int *packets, int *nbytes)` -- devolve o número de mensagens na fila em `packets` e o tamanho total das mensagens em `nbytes`.

Produto escalar (troca de mensagens) 1/2

```
#include "BspLib.hpp"
int main(int argc, char **argv) {
  int pid, nprocs = 4;
  double produto;
  double *listaProdutos = (double *) malloc(nprocs *
                                            sizeof(double));
  bsp_begin (nprocs);
  pid = bsp_pid ();
  produto = calculaProdutoLocal (pid, nprocs);
  int tagsize = sizeof(int);
  bsp_set_tagsize( &tagsize );
  // envio do produto escalar local a todos os outros
  for (int proc = 0; proc < bsp_nprocs( ); proc++)
    bsp_send (proc, &pid, &produto, sizeof(double));
  bsp_sync ( );
```

Produto escalar (troca de mensagens) 2/2

```
  for (int proc = 0; proc < bsp_nprocs( ); proc++) {
    int messageSize, source;
    bsp_get_tag (&messageSize, &source)

    bsp_move( &listaProdutos[source], messageSize)
  }
  calculaProdutoEscalarFinal (listaProdutos);
  bsp_end( );
}
```

Modelo OpenMP

API proposta pela *OpenMP Architecture Review Board (ARB)* para sistemas com memória compartilhada.

Versão inicial proposta em 1997, atualmente versão 3.0 (maio 2008).

Modelo de execução é baseado em `fork-join`.

Não existem instruções específicas para troca de mensagens (a comunicação é baseada em memória compartilhada).

KAAPI

O *Kernel for Asynchronous and Adaptive Parallel Interface* permite o desenvolvimento e execução de aplicações distribuídas baseadas em fluxos de trabalho.
Começou a ser desenvolvida no projeto MOAIS do LIG/INRIA em 2005, e é baseada em bibiotecas anteriores do mesmo grupo.

O escalonamento das tarefas aos processadores alocados é realizado dinamicamente.

Utiliza um algoritmo do tipo roubo de trabalho (*work-stealing*).

Fornece tolerância a falhas através de *checkpoints* do estado do fluxo de trabalho (não dos processos em si).

Middleware de grades

- Sistemas conhecidos internacionalmente
  - Condor
  - Legion
  - Globus
  - gLite
- Alguns sistemas brasileiros
  - OurGrid
  - InteGrade
  - EasyGrid

Globus

- O middleware de grade mais conhecido atualmente.
- Ian Foster e Carl Kesselman
- Argonne National Lab., University of Chicago, NCSA, University of Illinois.
- Suas origens remontam o I-WAY
  - grade temporária composta por 17 instituições
  - apresentado no Supercomputing'95

Evolução do Globus

- versão 1.0: 1998 (uso interno)
  versão 2.0: 2002 (disseminação mundial)
- versão 3.0: 2003 (comitê de padronização)
- versão atual: 4.0.5 pode ser baixada como software livre de www.globus.org

- Padrão OGSA (*Open Grid Services Architecture*)
  - conjunto de Web Services definidos pelo Open Grid Forum.

Globus

- O middleware de grade mais conhecido atualmente
  - Pode ser baixado gratuitamento (Versão 4.0.5)
  - Padrão OGSA (Open Grid Services Architecture)
- Utiliza o conceito de *Grid Services*
  - Comunicação baseada em Web Services
    - Baixo acoplamento entre os recursos
    - Desempenho ruim
  - Adiciona funcionalidades a Web Services
    - Descoberta de serviços
    - Serviços nomeados
    - Serviços com estado

Serviços do GT4

Serviços do Globus Toolkit 4:

- Monitoring and Discovery Service (MDS)
  - *Trigger*: coleta dados de uso de recursos e dispara um gatilho quando certas condições são satisfeitas.
  - *Index*: agrega em um único índice centralizado um conjunto de informações sobre recursos espalhados pela rede
  -
- Grid Resource Allocation and Management (GRAM)
  - responsável por localizar, submeter, monitorar e cancelar a execução de tarefas

Serviços do GT4

- GridFTP
  - permite a distribuição de arquivos na grade
- Replica Location Service
  - registro e recuperação de réplicas
  - mapeamento entre nomes lógicos e físicos
- Segurança
  - credenciais X.509 para identificação de usuários, serviços e recursos
  - SSL para comunicação segura
  - OpenSAML para gerenciamento de autenticação, atributos e autorizações.

Serviços do GT4

- Bibliotecas de tempo de execução formam o *Common Runtime*
  - XIO para entrada e saída usando vários protocolos
  - C Common Libraries unifica tipos e estruturas de dados e chamadas ao sistema operacional
  - bibliotecas de suporte a C, Java e Python.

OurGrid

- Um dos principais projetos brasileiros da área.

- Universidade Federal de Campina Grande.
- Parceria e financiamento HP Labs.

- Middleware Java para execução de aplicações seqüenciais do tipo saco de tarefas (*bag-of-tasks*)

OurGrid

- Principal motivação:
  - mecanismo simples para instalação e configuração de grades
  - (infra-estruturas mais usadas, p.ex., Globus, demandam um grande esforço de configuração)
- Para tanto desenvolveu-se um mecanismo simples baseados em redes par-a-par (P2P) para estruturação da federação de aglomerados.

OurGrid: redes de favores

- Usa o conceito de “rede de favores” para determinar a prioridade dos usuários na utilização de recursos.
- Quem oferece muitos recursos à Grade, pode utilizar mais recursos da Grade.

- Essa abordagem facilita o gerenciamento, evitando configurações manuais.

OurGrid: características

- Usa Xen para prover segurança mas não se preocupa com privacidade ou criptografia.
- Intencionalmente, não possui um escalonador global. O agente pessoal MyGrid é responsável pelo escalonamento das aplicações de um usuário.
- Está em produção desde dezembro de 2004 e reúne dezenas de laboratórios no Brasil e no exterior.

Arquitetura do OurGrid

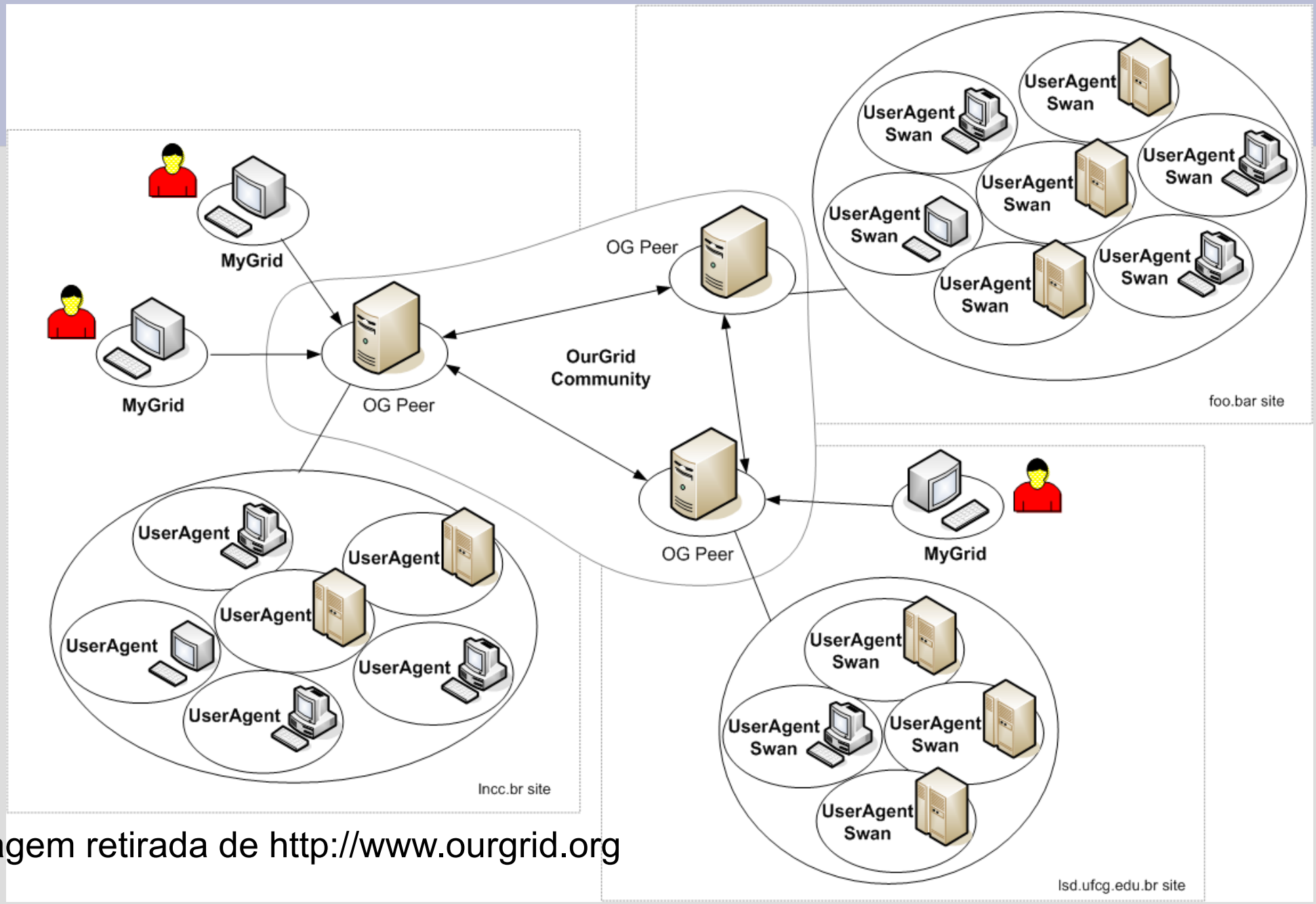


Imagem retirada de http://www.ourgrid.org

Grades Oportunistas

- Foco na utilização ciclos computacionais ociosos estações de trabalho compartilhadas
  - Máquinas de laboratórios, professores, etc.
  - Deve manter a Qualidade de Serviço dos donos das máquinas compartilhadas
  - Máquina passa do estado ocioso para utilizada
    - Aplicações da grade devem ser migradas para outras máquinas

Grades Oportunistas

- Vantagens
  - ✔ Baixo custo: utiliza hardware já existente
  - ✔ Economia de recursos naturais
    - ✔ Eletricidade, refrigeração, espaço físico
- Desvantagens
  - ✗ Gerenciamento de recursos é mais complicado
  - ✗ Menor desempenho: Utiliza apenas períodos ociosos das máquinas

Exemplo: SETI@HOME

- Sistema de computação distribuída
  - Análise de sinais de rádio-telescópios
  - Objetivo é por procurar por padrões não-naturais nos sinais obtidos
- Quebra o problema em blocos independentes
  - Não existe comunicação entre os nós
- Distribui grupos de blocos a cada computador
  - Computador realiza análise dos dados contidos nos blocos recebidos
  - Dados são enviados de volta a um servidor central

Exemplo: SETI@HOME

- Ótimo exemplo do poder computacional
  - Desempenho total até Janeiro de 2006
    - 7.745.000.000.000 Gflops
    - Mais de 5 milhões de usuários
  - Core 2 Duo (2.16GHz): ~ 3 GFlop/s
    - 259.200 Gflops por dia
  - Computador mais rápido do mundo até 2007:
    - Blue Gene/L (212.992 processadores)
    - 478.200 GFlop/s
    - 187 dias no Blue Gene/L ↔ Seti@Home
  - Hoje: Linux Roadrunner da IBM
    - 122.400 núcleos; pico: 1.375.776 GFlop/s

Exemplo: Folding@home

- Folding@home
  - Sistema para realizar simulações computacionais do enovelamento de proteínas
  - Aplicações: cura de doenças, como o câncer, Alzheimer, Parkinson, etc.
  - Possui versão que pode ser executada em Playstation 3.

Condor: um sistema pioneiro

- Grade oportunista
  - Desenvolvido na *University of Wisconsin-Madison* desde o final da década de 1980.
  - Permite utilizar ciclos ociosos de estações de trabalho compartilhadas.
- Condor *pools*: grupos de computadores
  - Condor *pools* podem ser conectados entre si
- Hoje em dia possui suporte a aplicações seqüencias, saco de tarefas e MPI.

Exemplo: Condor

- Computadores fazem anúncios dos recursos disponibilizados
  - Estes recursos são então monitorados pelo Condor.
- Aplicação é submetida para execução
  - *Match-maker* repassa a execução a máquinas que possuam os recursos necessários para executar a aplicação.
- Caso uma máquina seja requisitada pelo dono
  - Aplicação é migrada para outro nó.
  - Limitação: aplicações paralelas rodam em nós dedicados.

InteGrade: middleware para grades oportunistas

- Iniciativa de 6 universidades brasileiras
  - IME-USP, PUC-Rio, UFMS, UFG, UFMA e UPE
- Desenvolvido colaborativamente com o software livre disponível em www.integrade.org.br
- Implementado em Java, C++ e Lua.
- Comunicação baseada em CORBA.
- Suporte a aplicações C, C++, Fortran e Java.

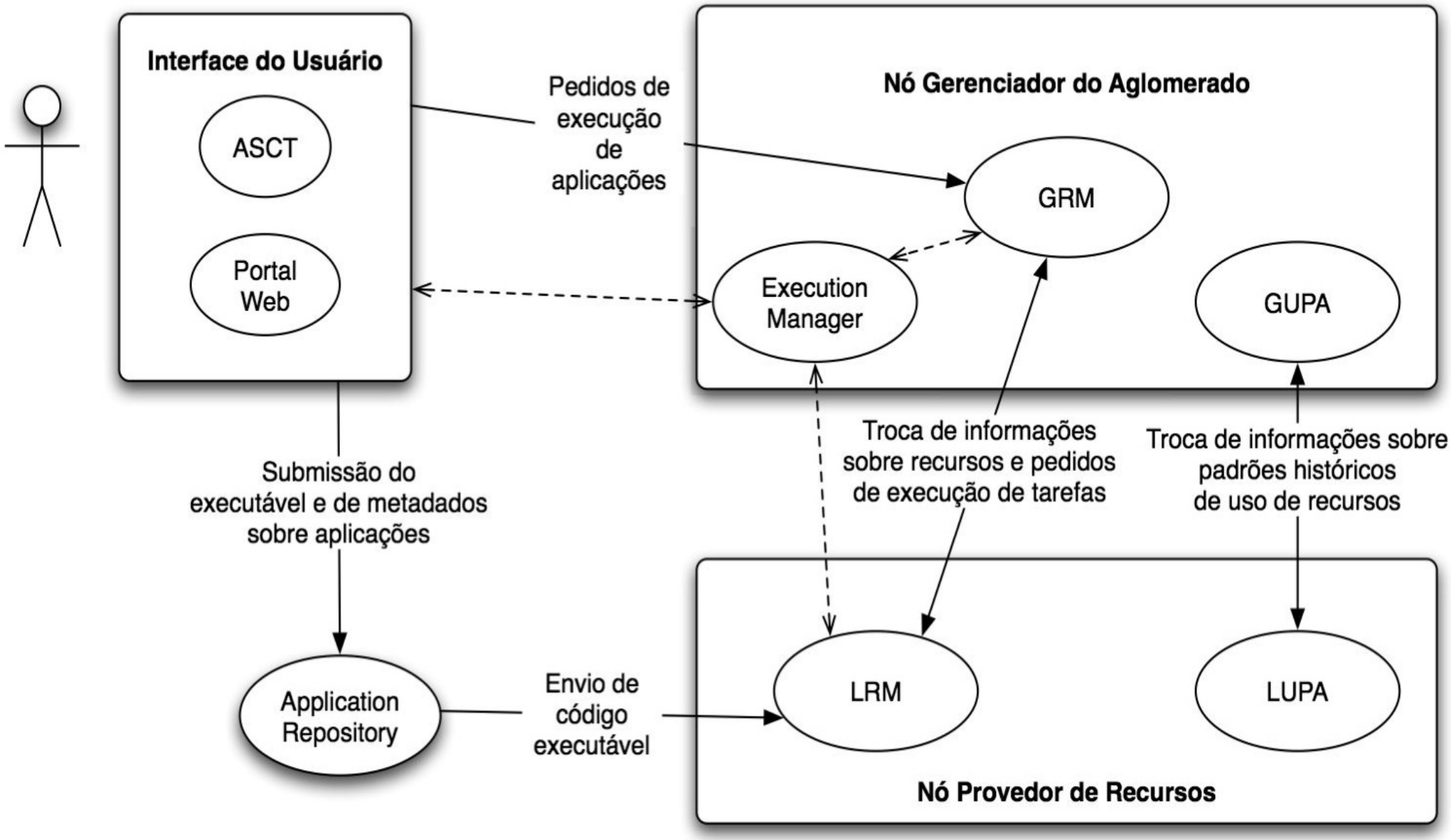
Motivação

- Desenvolver um middleware que permita utilizar recursos ociosos de máquinas compartilhadas já existentes nas instituições.
- Objetivo é realizar a execução de aplicações paralelas computacionalmente pesadas.
- Armazenamento de dados de aplicações no espaço livre em disco das máquinas compartilhadas.
- Importante para instituições com recursos financeiros escassos.
- Ambientalmente mais responsável.

Características

- Foco na utilização de computadores pessoais
- Suporte a aplicações seqüenciais e paralelas
  - Saco de tarefas, BSP e MPI
- Portal Web para submissão de aplicações
- Armazenamento distribuído de dados
- Em desenvolvimento:
  - Preservação da Qualidade de Serviço dos donos de máquinas compartilhadas
  - Segurança baseada em redes de confiança

Arquitetura



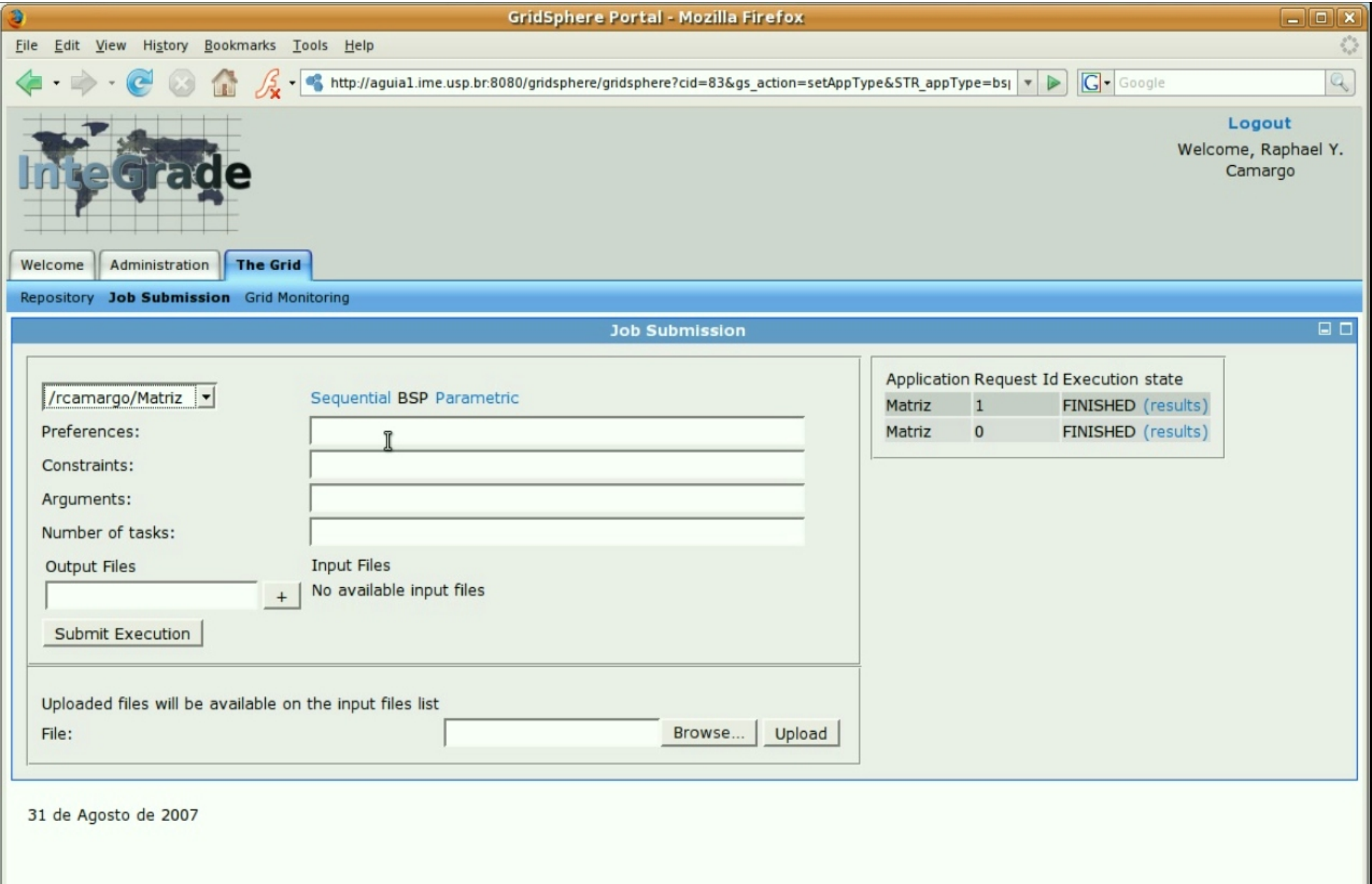
Principais componentes

- Global Resource Manager (GRM)
  - gerencia os recursos de um aglomerado
  - seleciona em quais máquinas cada aplicação submetida será executada
- Local Resource Manager (LRM)
  - gerencia uma máquina provedora de recursos
  - inicia a execução de aplicações na máquina
- Application Repository (AR)
  - armazena os executáveis das aplicações dos usuários

Principais componentes

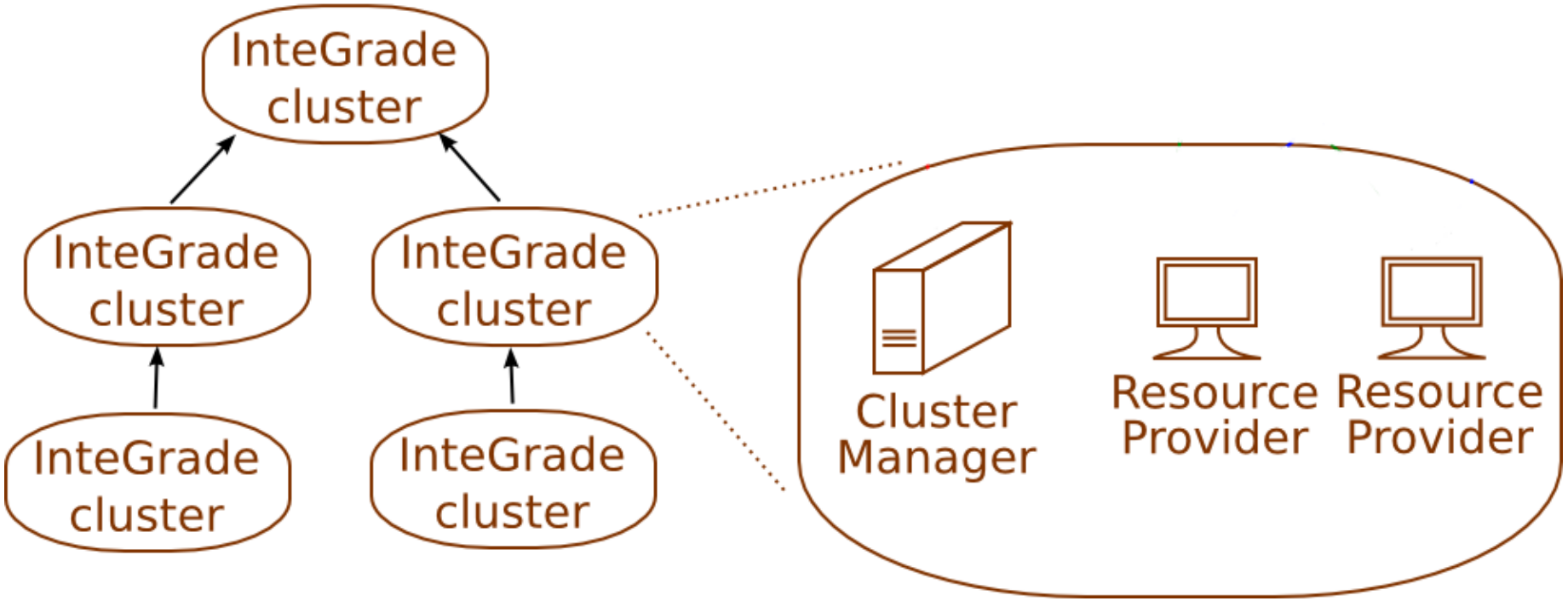
- Application Submission and Control Tool (ASCT)
  - permite a submissão de aplicações e a obtenção e visualização de resultados
- Portal Web
  - versão Web da ferramenta ASCT
- Local Usage Pattern Analyser (LUPA)
  - analisa o padrão de uso das máquinas da grade
  - permite ao escalonador realizar melhores escolhas no momento de definir máquinas que executarão uma aplicação

Portal InteGrade

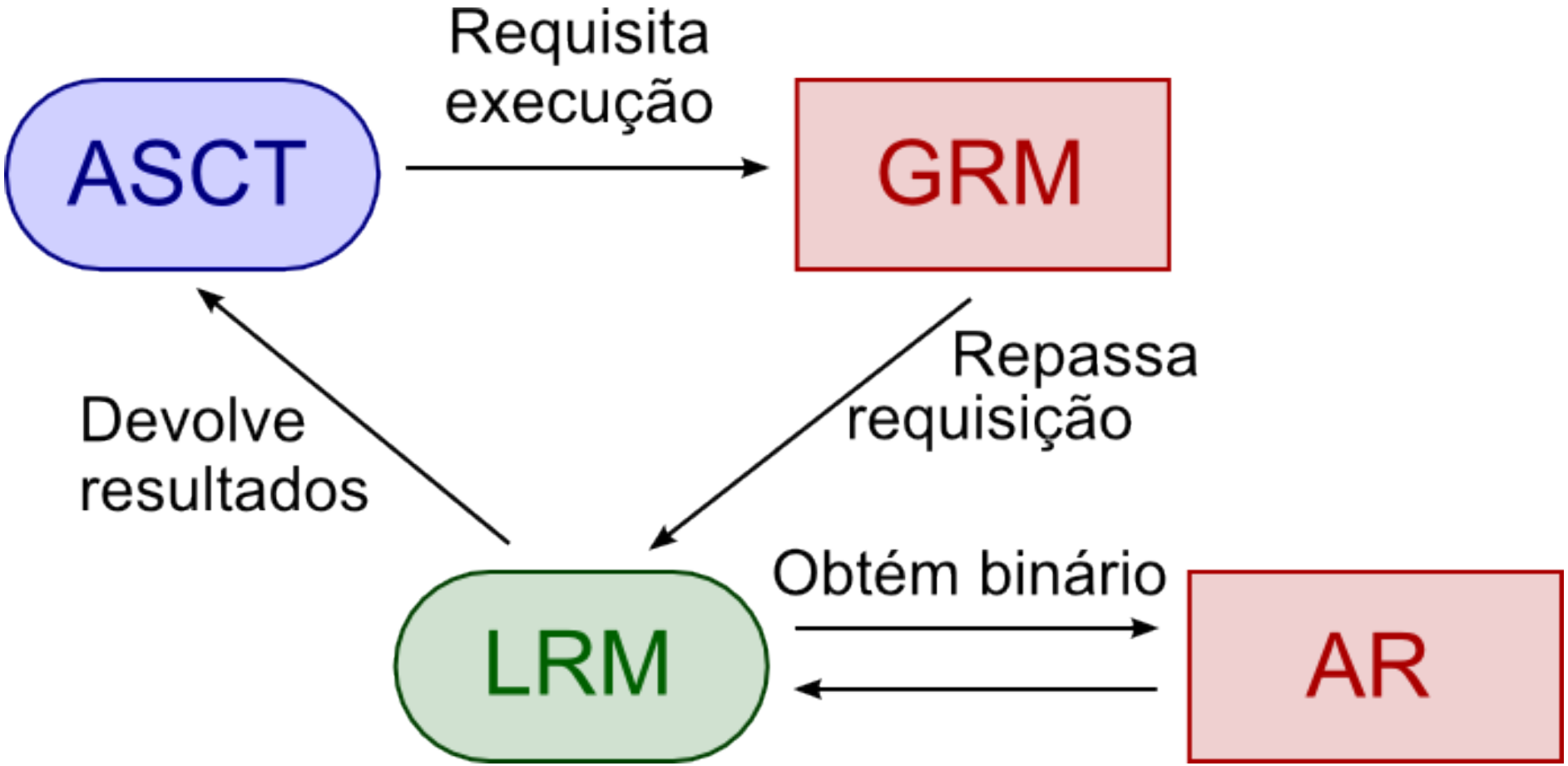


Federação de Aglomerados

- Federação de aglomerados
  - Aglomerados conectados em estrutura hierárquica
- Aglomerados possuem informações aproximadas sobre recursos disponíveis em aglomerados filhos



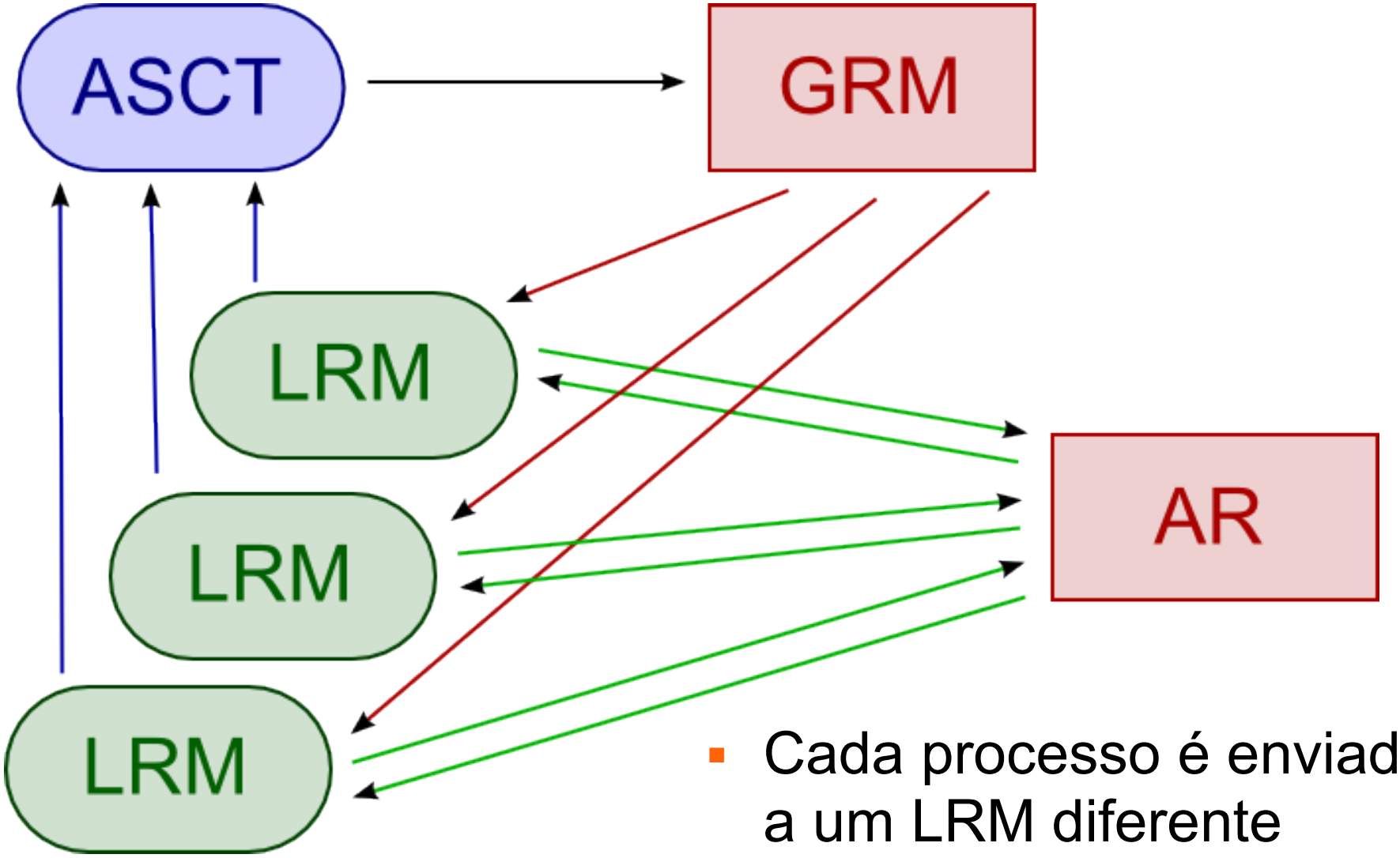
Execução de Aplicações



Execução de Aplicações

1. Usuário submete aplicações através do Portal
   - Pode ser executado a partir de qualquer máquina
2. GRM determina em quais máquinas a aplicação será executada (escalonamento)
3. Requisição é repassada para as máquinas selecionadas
4. Máquinas obtêm dados de entrada e os binários da aplicação e realizam sua execução
5. Após o término da execução, arquivos de saída são enviados à máquina executando o portal

Execução de Aplicações Paralelas



- Cada processo é enviado a um LRM diferente

Execução Robusta de Aplicações

- Aplicações executadas durante ciclos ociosos de máquinas não-dedicadas
  - Podem ficar indisponíveis ou mudar de ociosa para ocupada inesperadamente
  - Execução da aplicação é comprometida
    - É preciso reiniciar a execução do início
- Mecanismo de tolerância a falhas
  - Reinicia automaticamente a aplicação de um ponto intermediário de sua execução
  - É totalmente transparente ao usuário

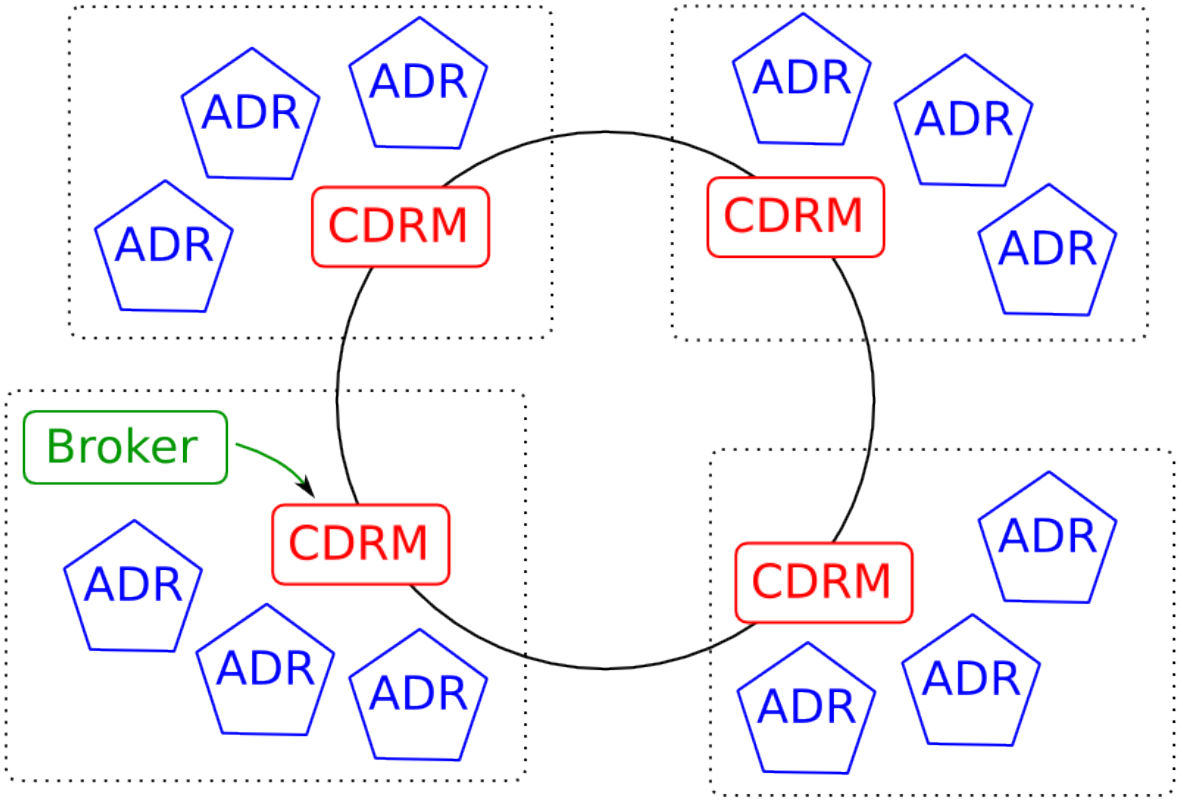
Armazenamento Distribuído

- Aplicações da grade podem utilizar ou produzir grandes quantidades de dados
  - Dados de aplicações podem ser compartilhados
- Abordagem tradicional: servidores dedicados
- Grade oportunista
  - Máquinas com grandes quantidade de espaço livre
  - Ambiente altamente dinâmico
    - Composto por dezenas de milhares de máquinas
    - Utilizar somente períodos ociosos
    - Máquina entram e saem do sistema continuamente

OppStore

- Middleware que permite o armazenamento distribuído de dados da grade
  - Provê armazenamento confiável e eficiente
  - Utiliza o espaço livre de máquinas compartilhadas
- Organizado como federação de aglomerados
  - Similar à maioria das grades oportunistas
    - Aglomerados do OppStore são mapeados em aglomerados da grade oportunista
  - Facilita o gerenciamento do dinamismo do sistema
  - Aglomerados conectados por uma rede par-a-par

Arquitetura do OppStore



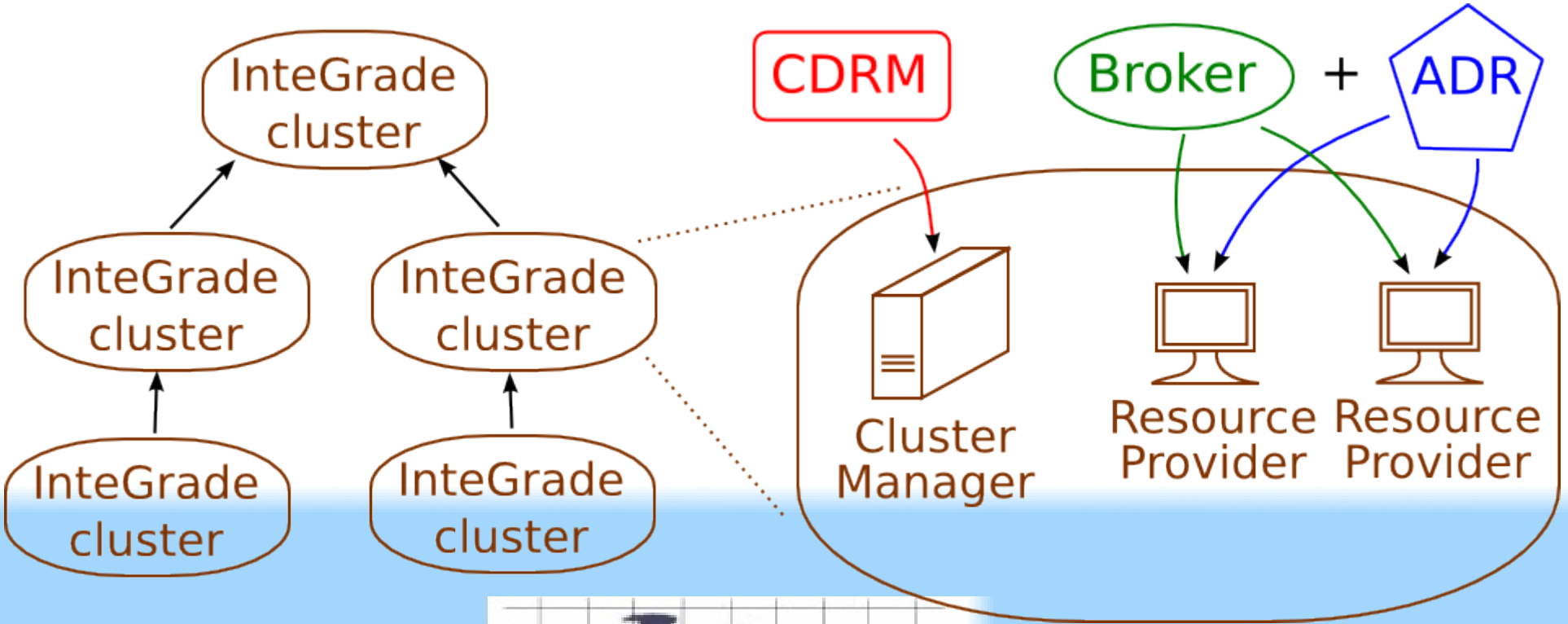
- Repositório Autônomo de Dados (ADR)
- Gerenciador de Repositórios de Dados (CDRM)
- Intermediador de Acesso (Access Broker)

Armazenamento de arquivos

- Arquivos codificados em fragmentos redundantes
  - Fragmentos armazenados em aglomerados distintos
- Vantagens
  - Maior tolerância a falhas
  - Bom desempenho, pois fragmentos são transferidos em paralelo

Implantação sobre o InteGrade

- CDRM → Máquina gerenciadora do aglomerado
- ADRs e Brokers → Provedores de recursos
- Interface com o InteGrade em desenvolvimento



Estado Atual do InteGrade

- Atualmente na versão 0.4.
- Execução em 2 aglomerados e 2 laboratórios compartilhados do IME-USP e em aglomerados de outras universidades.
- Permite execução de aplicações paralelas
  - MPI, BSP e paramétricas (saco de tarefas)
- Precisam ser realizados mais testes
  - Para tal, precisa-se de mais usuários e aplicações
- Contato:
  - integrade-support@integrade.org.br
  - www.integrade.org.br

Grades Computacionais:

Conceitos Fundamentais e Casos Concretos

Grades em Funcionamento
Pesquisa Atual em Grades

Fabio Kon, Alfredo Goldman

Universidade de São Paulo
{kon,gold}@ime.usp.br

SBC - JAI - Belém - 15 e 16/07/08

Roteiro

Preliminares

Grid 5000

PlanetLab

EGEE

Pesquisa atual em grades

Grades em produção

Atualmente existem inúmeras grades em funcionamento.

Escolhemos três tipos para ilustrar as possibilidades :

- acadêmica
- planetária
- alto desempenho

Mas, existem outras, principalmente baseadas em Globus, como TeraGrid e Open Science Grid.

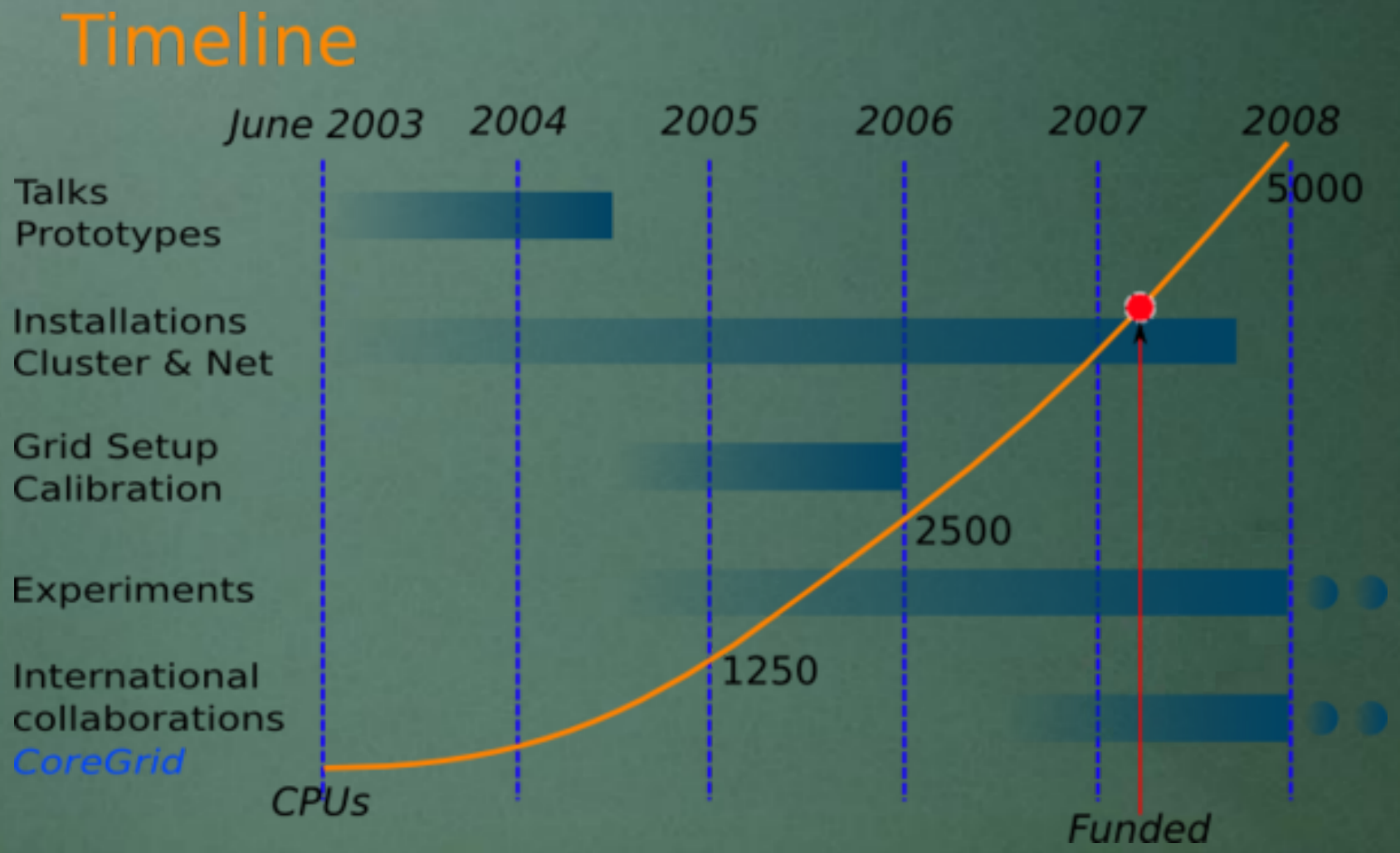
Grid 5000

Grade francesa para pesquisas em Ciência da Computação.

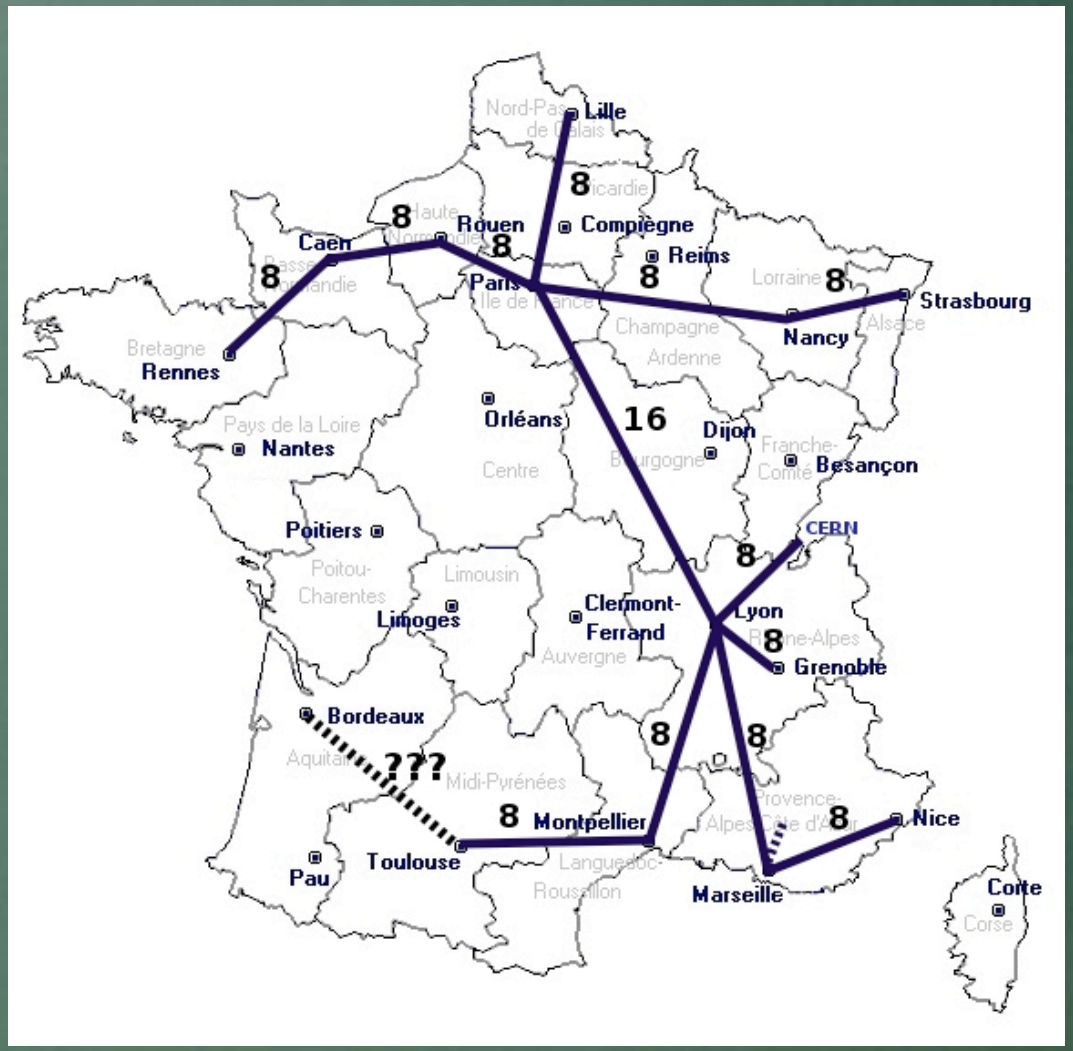
Características principais :

- INRIA, CNRS e vários outros orgãos financiadores
- composta por 9 aglomerados na França
- interligada por rede rápida e dedicada
- cada aglomerado tem de 256 a 1000 processadores
- ambiente estanque
  1. experimentos reprodutíveis
  2. segurança
  3. conexões completamente conhecidas

Grid 5000 evolução



Grid 5000 geograficamente



Processadores da Grid 5000

Proc/Sites	Nancy	Bordeaux	Grenoble	Lille	Lyon
Intel Xeon				92	
AMD Opteron	94	322		198	260
Itanium 2			206		
Xeon EM64T	240	102			
Xeon IA32			64		
total	334	424	270	290	260

Proc/Sites	Orsay	Rennes	Sophia	Toulouse	total
Intel Xeon					92
AMD Opteron	684	326	356	276	2516
Itanium 2					206
Xeon EM64T		198			540
Xeon IA32					64
total	684	524	356	276	3418

Características

Ambiente heterogêneo

processadores AMD Opteron, Intel Itanium, Intel Xeon e PowerPC

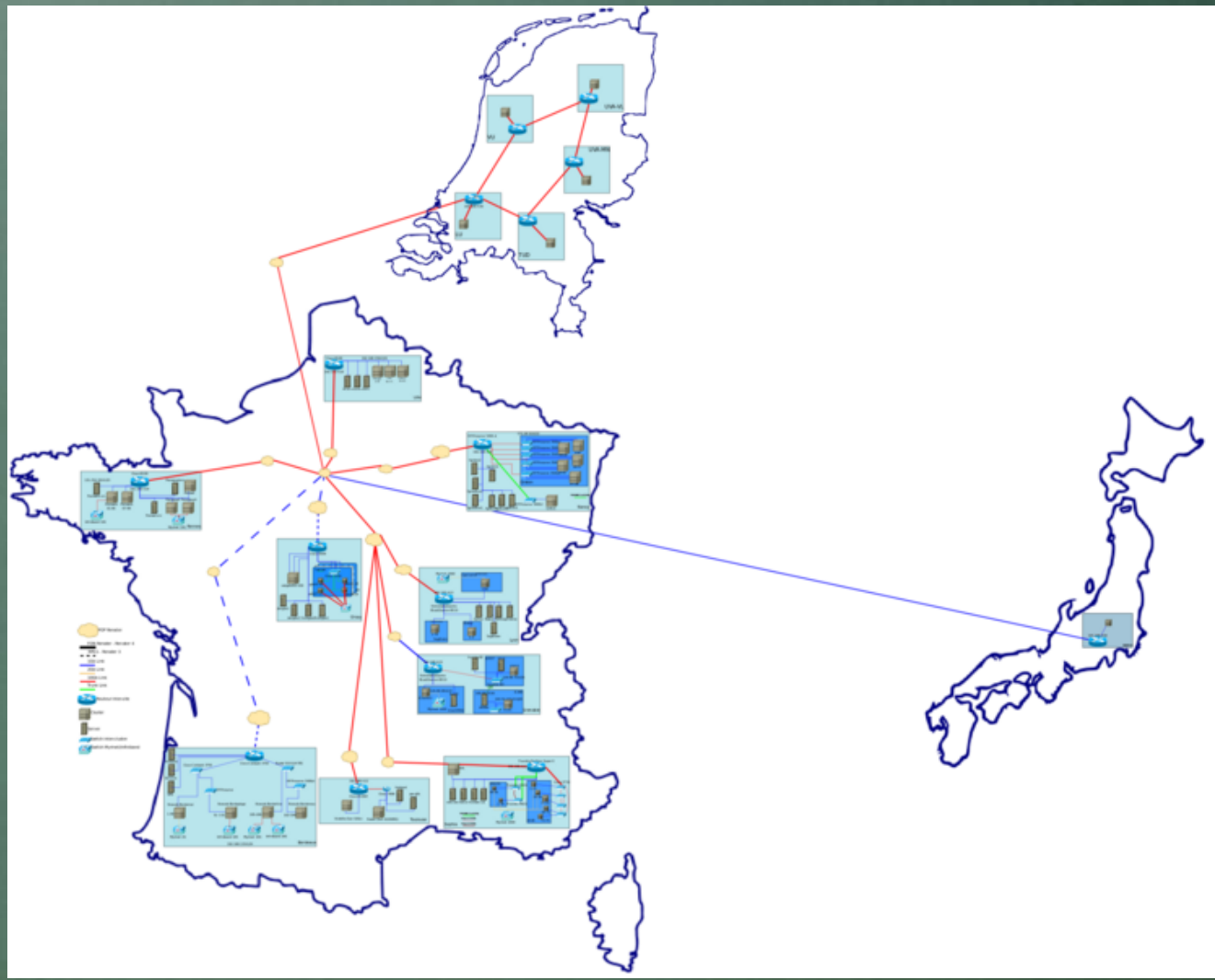
rede de interconexão Myrinet 2G, Myrinet 10G e InfiniBand

sistema operacional completamente configurável
O usuário pode escolher a imagem para inicializar as máquinas.

Várias ferramentas :

- injeção automática de falhas
- injeção de tráfego na rede
- medida de consumo de energia (início 7/08)

Grid 5000 e suas conexões

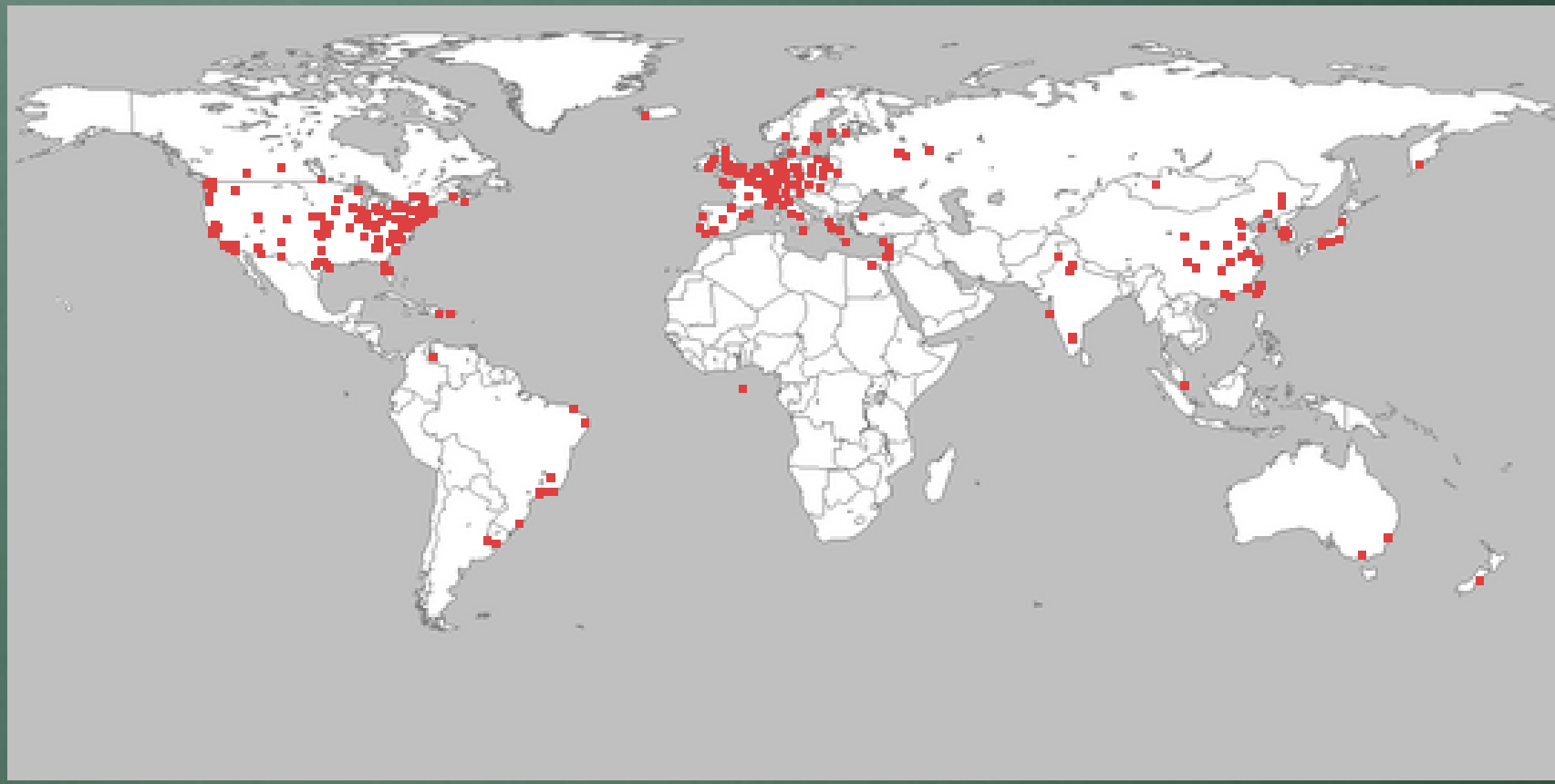


PlanetLab

Grade com máquinas em todo o planeta.

- Projeto iniciado em 2002 e gerenciado pelas universidades de Princeton, Berkeley e Washington.
- Objetivo principal : desenvolvimento de serviços de rede.
- Conta com 895 nós em 461 localidades (7/08).
- Conexão atraves da Internet pública.
- Todas as máquinas executam o mesmo pacote de software.
  1. Mecanismos para reiniciar máquinas e distribuir atualizações.
  2. Ferramentas para monitorar os nós.
  3. Gerenciamento de contas de usuários.
  4. Distribuição de chaves.

PlanetLab no mundo



Enabling Grids for E-science

- Financiado pela união européia
- Sucessor do Data-Grid
- EGEE I (março 04) encerrado, EGEE II(abril 06)
- Grade para aplicações científicas
- Composta por 68 mil processadores, em 250 sítios de 48 países (7/08)
- 8 mil usuários, processa mais de 150 mil aplicações por dia
- 20 PetaBytes de armazenamento
- Disponível permanentemente
- Usa o middleware gLite

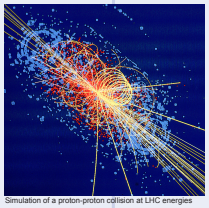
Aplicações na EGEE

- Física de alta energia
  1. maior utilização
  2. vários projetos ligados ao *Large Hadron Collider*
- Física de particulas espaciais
- Fusão
- Observatório da grade (coleta da dinâmica e do uso da EGEE)
- Ciências da terra (principalmente geologia)
- Ciências da vida (imagens médicas, bioinfomática, descoberta de drogas)

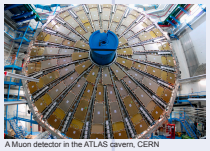
EGEE Applications

Enabling Grids for E-sciencE

High-Energy Physics



Simulation of a proton-proton collision at LHC energies



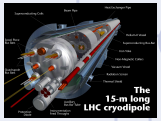
A Muon detector in the ATLAS cavern, CERN

The High-Energy Physics (HEP) community is one of the pilot application domains in EGEE, and is the largest user of its grid infrastructure.

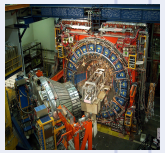
At present, the major users are the four experiments (ALICE, ATLAS, CMS and LHCb) of the Large Hadron Collider (LHC), which will begin with the first proton-proton collisions in autumn 2008 and achieve the design luminosity in 2010.

These four experiments are using grid resources for large-scale production work involving more than 150,000 jobs/day on the EGEE infrastructure and in collaboration with its sister projects OSG in the USA and NDGF in the Nordic countries.

Other major HEP experiments, such as BaBar, CDF, DØ, H1 and ZEUS have also adopted grid technologies and use the EGEE infrastructure for routine physics data processing.

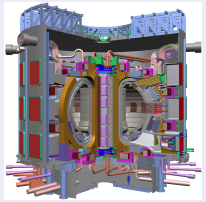


A magnet section from the 27km long LHC ring



The Collider Detector at Fermilab (CDF)

Fusion



A cutaway of the proposed ITER fusion reactor (Published with the permission of ITER)

Commercial exploitation of fusion energy still needs to solve several outstanding problems, some of which require a strong computing capacity. The International Thermo nuclear Experimental Reactor (ITER), a joint international research and development project, aims to demonstrate the scientific and technical feasibility of fusion power and could potentially produce 500 MW of power by 2016. The exploitation of ITER requires a modelling capability that is at the limit of the present state of the art. Therefore, computing grids and high performance computers are basic tools for fusion research.
Presently several applications are already running on the EGEE grid, namely Massive Ray Tracing, Global Kinetic Transport and Stellarator optimisation, that have helped to open new avenues of research. A number of new applications devoted to ITER simulation will be ported to the grid in close collaboration with EUFORIA project. Data management in large international experiments and the development of complex workflows are the activities that will complement grid computing.

Astro-Particle Physics

The community currently includes 17 institutes, all contributing with applications ported to EGEE. The most relevant among them are Planck, MAGIC, SWIFT/MERCURY, and LOFAR. All of them share problems of computation involving large-scale data acquisition, simulation, data storage, and data retrieval that the grid helps to resolve. Planck and MAGIC have been in EGEE since 2004. The ESA Planck satellite, to be launched in 2008, will map the sky using microwaves, with an unprecedented combination of sky and frequency coverage, accuracy, stability and sensitivity. The MAGIC telescope, on the island of La Palma in the Canary Islands, is an imaging atmospheric Cherenkov telescope that has been in operation since late 2004.

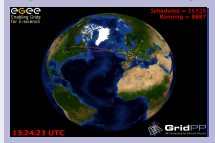
Artist's rendition of the ESA Planck Satellite (Copyright ESA 2002-Illustration Medialab)

The MAGIC telescope, La Palma

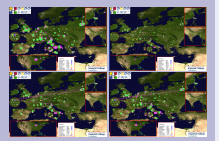
EGEE Applications and their support

The Enabling Grids for E-sciencE (EGEE) project began by working with two scientific groups, High Energy Physics (HEP) and Life Sciences, and has since grown to support formally astronomy, astrophysics, computational chemistry, earth sciences, fusion, and computer science. The full user community runs applications from research domains as diverse as multimedia, finance, archaeology, and civil protection. Researchers in these areas collaborate through Virtual Organisations (VOs) that allow them to share computing resources, common datasets, and expertise via the EGEE grid infrastructure. End users can join existing VOs or create new VOs tailored to their needs. Those with existing computing resources can also federate them with the EGEE grid infrastructure to facilitate load balancing with other users and groups.

To help the user community take advantage of the benefits of grid computing, EGEE provides a range of support services to its users: direct user support, Virtual Organisation (VO) support, and application porting support. Through other activities, the project also provides beginner and expert training on various topics



EGEE partner GridPP also produces a 3D (left) and 2D (right) visualisation of the infrastructure. Both are available from http://gridportal.hep.ph.ic.ac.uk/rtm/.



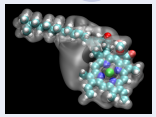
Getting Involved

Details of how to join EGEE can be found on the User & Application portal at http://egeena4.lal.in2p3.fr/

Members of business and industry are also encouraged to join the project.
EGEE runs an Industry Forum, hosts special Industry Days and works with industrial applications through an Industry Task Force, as well as collaboration with the commercial sector in the framework of the EGEE Business Associate Programme.
For more details see the "EGEE and Business" section on www.eu-egee.org.

Computational Chemistry

The Computational Chemistry and Gaussian virtual organizations were established to allow access to chemical software packages on the EGEE infrastructure. At present both freely available (GAMESS, COLUMBUS, DL_POLY, RWAVEP or ABCtraj) and commercial software packages, including Gaussian, Turbomole and Wien2K, are used by chemists to understand better molecular properties, to model chemical reactions or to design new materials. The availability of chemical software is also beneficial for other communities as a source of molecular data parameters for their simulations.



Grid Observatory

The application part of EGEE includes the Grid Observatory. Its aim is to develop a scientific view of the dynamics of grid behaviour and usage. The Grid Observatory will make available traces of users and middleware activity on the grid for study by computer scientists, based on the existing extensive monitoring facilities. These data will excite researchers in both the grid and machine learning areas and improve the connection between the EGEE project and computer science, with a specific impact sought to the emerging field of autonomic computing. Modelling the dynamics of both the grid and the user community will contribute to a better understanding of the areas of dimensioning and capacity planning, to performance evaluation, and to the design of self-regulation and maintenance functionalities such as real-time fault diagnosis. Eventually, the improved understanding of grid activity may improve the reliability, stability, and performance of the grid itself.

Earth Sciences

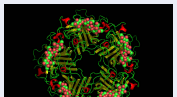
EGEE supports two related communities in the area of Earth Sciences: Research (ESR) and Geosciences (EGEODE).

The applications of the ESR domain cover various disciplines. The most numerous applications are in seismology with the re-analysis of the whole GEOSCOPE data set, the determination of the earthquake characteristics a few hours after the data arrival and numerical simulations of wave propagation in complex 3D geological models. Several applications are based on atmospheric modelling like the long-range air pollution transport over Europe, the regional climate, and the ozone index calculations.
In hydrology, applications include flood forecasting and the calculation of sea water intrusion into coastal aquifers, both related to risk management. Other applications are related to geomorphology, meteorology, or planetol-

Life Sciences

The life sciences are a major application area for the EGEE project and have been used to guide the implementation of the infrastructure from the start. With more than 30 applications deployed and being ported, the domain had more than 200,000 jobs executed per month in 2007.

- The medical imaging domain works on a number of related systems, many of them in the compute-intensive area of image registration. This enables techniques such as 'virtual biopsies' for cancer diagnosis that avoid invasive surgical procedures.
- The bioinformatics domain studies genes, proteins



Pesquisa atual em Grades 1/3

Abaixo listamos alguns dos tópicos « quentes » em pesquisa sobre grades :

1. Contabilidade e Economia de Grade
   - Como controlar a utilização dos recursos computacionais pelos usuários;
   - Garantir justiça no compartilhamento;
   - Pagamento pela utilização de recursos;
   - Mercado de compra e venda de recursos.
2. Grades autônomas
   - Mecanismos adaptativos para gerenciamento e QoS;
   - Mecanismos para auto-otimização, auto-proteção, configuração automática e tolerância a falhas automática.
   - Segurança de fácil administração (auto-gerenciamento).

Pesquisa atual em grades 2/3

3. Protocolos e algoritmos inteligentes para federação de aglomerados e escalonamento global
   - O que fazer quando a visão global do sistema não é possível ?
   - O uso único e informações locais podem levar ao desperdício.
   - Como maximizar a utilização dos recursos neste contexto ?

4. Escalonamento em grades
   - Buscar soluções que forneçam melhoras para todos os participantes.
   - Para isso são necessárias regras de conduta.
   - Atualmente existem duas linhas principais : rede de favores e teoria de jogos.

Pesquisa atual em Grades 3/3

3. Melhor aproveitamento das novas arquiteturas de hardware.
   - Arquiteturas multi-processadas são uma realidade.
   - Placas gráficas (GPUs) tem enormes capacidades de processamento.
   - Como aproveitar melhor estes novos recursos em grades ?

Onde obter mais informações?

1. www.integrade.org.br Nosso projeto :-)
2. www.ourgrid.org Outra iniciativa de origem nacional
3. www.globus.org The Globus Alliance
4. www.grid5000.fr Grid'5000
5. www.planet-lab.org PlanetLab
6. www.eu-egee.org Enabling Grids for E-sciencE
7. www.cs.wisc.edu/condor The Condor Project
8. www.legion.virginia.edu Legion : A Worldwide Virtual Computer
9. www.teragrid.org TeraGrid
10. www.opensciencegrid.org Open Science Grid
11. boinc.berkeley.edu Open-source software for volunteer computing and grid computing