

Alocação de processos a processadores

equilíbrio de carga e oportunismo



Distribuição de Carga

- carga = ?
 - processos a serem executados
 - processos em execução (migração)
 - dados descrevendo tarefas
- momento de distribuição
 - estático
 - dinâmico
 - » quão dinâmico?
 - » antes do início da aplicação inteira ou antes do início de cada processo
- quem distribui
 - sistema X aplicação



Distribuição de Carga

- estática
 - escalonamento de processos
 - grafos de precedência
- dinâmica
 - alocação de processos leva em conta estado do sistema
 - balanceamento?
- preemptiva
 - migração de processos para manter carga equilibrada
 - ou, em sistemas oportunistas, para devolver máquina a usuário local!



Quem coordena

- visão do sistema
 - melhor aproveitamento de recursos
- visão da aplicação
 - menor tempo total de execução

relação com
tipo de distribuição
realizada

equilíbrio X distribuição

visão do sistema: diferentes implementações!



passos

- monitoramento
- distribuição da informação
- decisões de alocação
- transferência de carga

» peso de cada uma dessas etapas sobre o sistema



monitoramento

- o que monitorar?
 - fila da CPU
 - atividade de E/S
 - uso de memória
 - ...
- estatísticas
 - médias
 - máximos
 - ...
- como configurar o que será monitorado?



distribuição de info

- sob demanda X periódicas
- info centralizada X distribuída



decisões de transferência

- visão central X visão distribuída (ou p2p)
- todos os processadores envolvidos?
 - sincronização?
- iniciada pelo receptor
 - carga baixa
- iniciada pelo emissor
 - carga alta

» flooding e swinging!!!



transferência de carga

- carga = processos ainda a serem iniciados
 - simples se sistema de arquivos central
 - necessidade de **orquestração** em caso contrário
- carga = processos em execução
 - complicado!
- carga = subconjunto de dados
 - problema central fica sendo o volume de dados transferido



exemplos - visão de sistema

- sistema operacional controlando conjunto de máquinas como recurso único
 - MOSIX
 - » hoje já um pouco diferente...

X

- aproveitamento de recursos ociosos
 - CONDOR

» em ambos os casos: migração de processos entre plataformas homogêneas



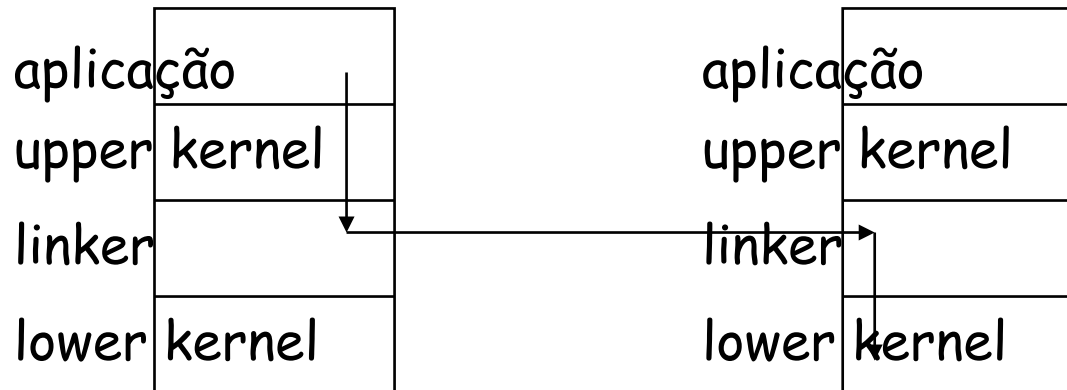
MOSIX

- <http://www.mosix.org/>
 - 1981
- Hebrew University, Jerusalem
- conceito de transparência de localidade
- kernel próprio com interface Unix
- migração de processos para equilíbrio de carga



MOSIX

- kernel implementa RPC
- chamadas a sistema transferidas para máquina apropriada



MOSIX: migração

- máquinas sobrecarregadas decidem destino de processos escolhidos
- carga medida por prontos e memória livre
- escolha de migração leva em conta
 - perfil (passado) do processo
 - tamanho -> tempo de migração
 - tempo de residência
 - i/o local e remoto
- objetivos:
 - maior throughput e menor tempo de resposta



cálculo de carga local

- fila dos prontos
- não apenas último valor, mas média de valores coletados desde o último cálculo de carga

$$\sum_{i=1}^n W_i$$

- essa medida é dividida por CPU_{speed}/CPU_{max} para normalização
- fator extra é adicionado para evitar swinging



infos mantidas em cada proc

- loadinfo:
 - número do processador
 - velocidade
 - carga conhecida
 - memória livre
- lista ordenada por idade da informação



disseminação: várias versões

- atualmente: envio de carga local e de terceiros
 - merge de infos recebidas com infos locais
 - arquitetura estilo *gossip*



decisões sobre migração

- automáticas:
 - tempo de resposta no destino
 - overhead de comunicação
 - » qtas msgs foram trocadas com processador destino?
 - tempo de migração
- explícitas
 - chamada de sistema migrate
 - disparo/suspensão de migração



depois da migração

- algumas chamadas de sistema são realizadas na máquina de execução e outras na máquina de submissão
 - transparência
 - alterações pequenas
- na máquina de submissão, estrutura de dados (home structure) mantém ponteiro para localização atual



programação paralela

- interface unix garante funcionamento de bibliotecas como pvm e mpi
- bibliotecas para programação mestre-escravo



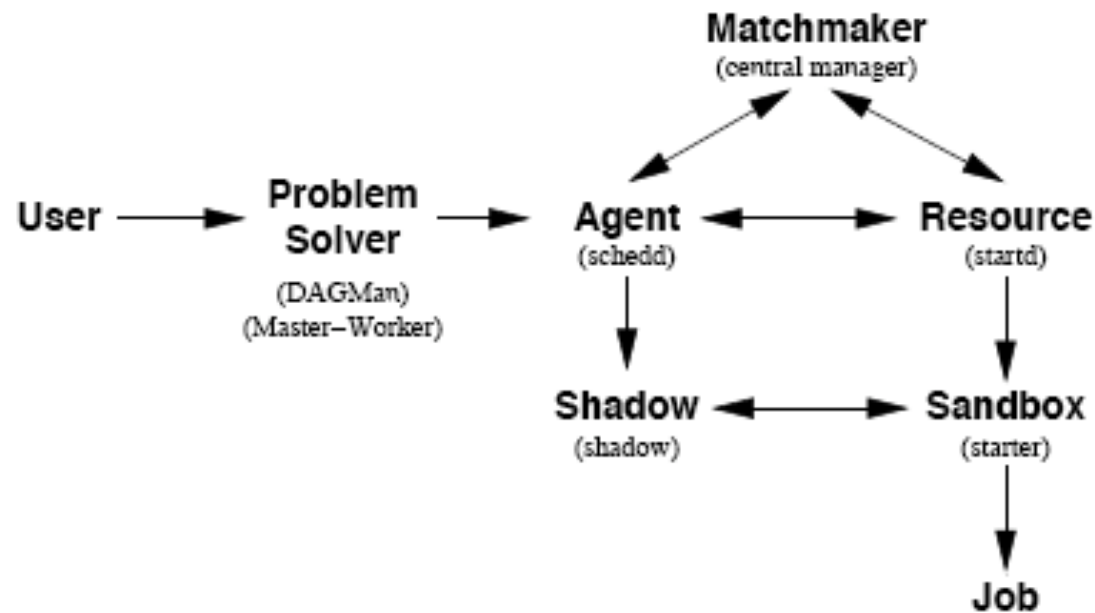
Condor - “oportunismo”

- <http://www.cs.wisc.edu/condor/>
 - 1988!
- “Leave the owner in control, regardless of the cost”
- sistema voltado para aproveitamento de recursos ociosos
- biblioteca CONDOR com wrappers:
 - chamadas de sistema redirecionadas para máquina origem
 - facilidades para segurança e para migração



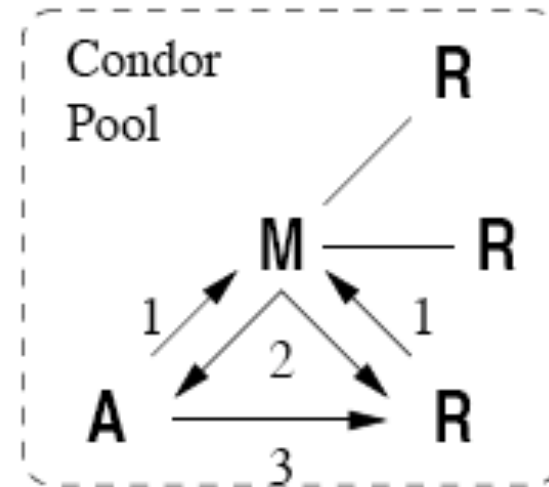
kernel Condor

- arquitetura básica com pequenas variações ao longo do tempo

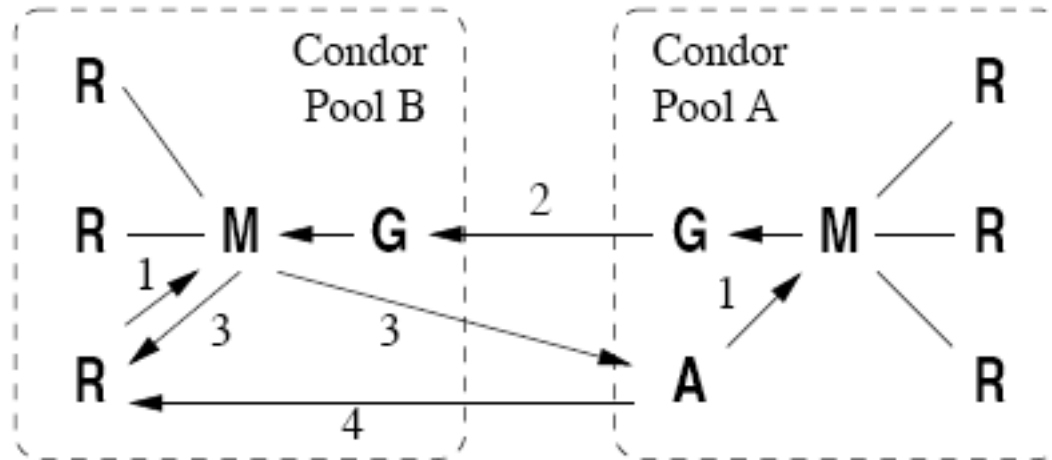


Condor - arquitetura inicial

- pool Condor ~1988
 - tipicamente máquinas em um único domínio administrativo
- agente entra com requisitos do usuário
- recurso entra com requisitos de dono da máquina
- matchmaker faz casamento e coloca requisitos coletivos



Condor - gateway flocking

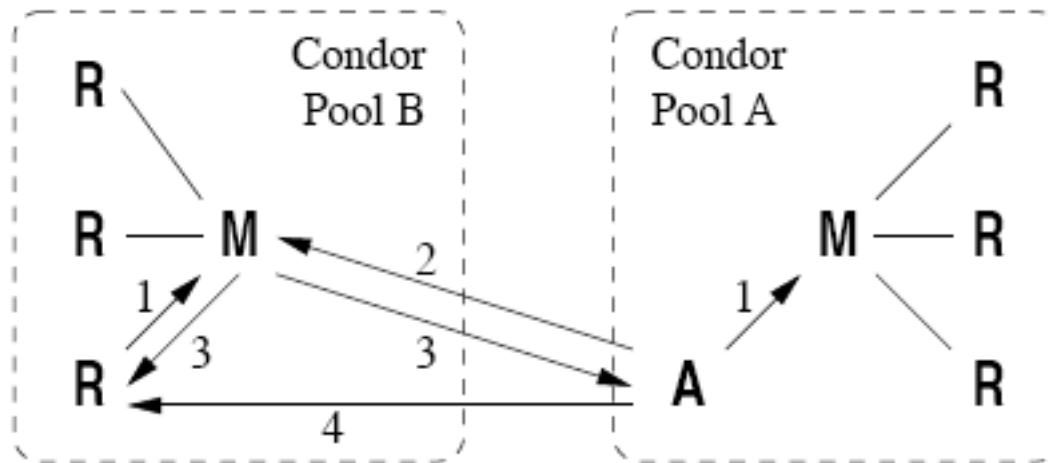


- cada pool é representado externamente por um único gateway
 - facilidades de transparência
 - problemas com contabilidade e reputação
 - adequado para parcerias institucionais



Condor - direct flocking

- um agente pode se comunicar com múltiplos matchmakers
 - acordos entre indivíduos e organizações



Matchmaking

Job ClassAd

```
[  
MyType = "Job"  
TargetType = "Machine"  
Requirements =  
((other.Arch=="INTEL" &&  
other.OpSys=="LINUX")  
&& other.Disk > my.DiskUsage)  
Rank = (Memory * 10000) + KFlops  
Cmd = "/home/tannenba/bin/sim-exe"  
Department = "CompSci"  
Owner = "tannenba"  
DiskUsage = 6000  
]
```

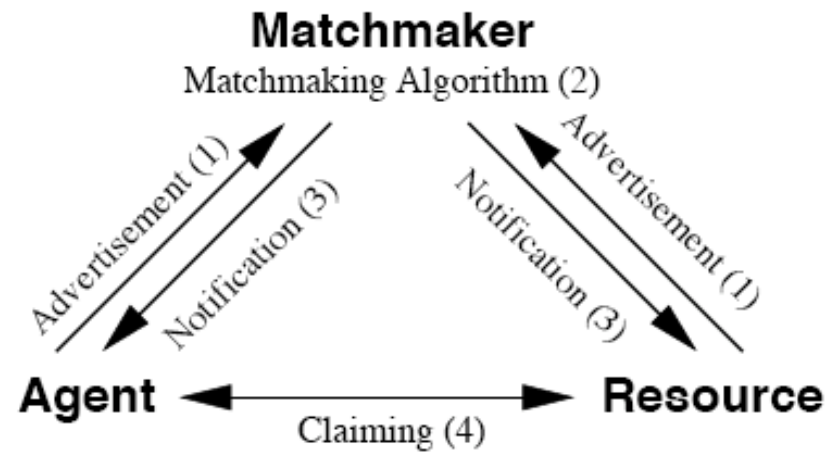
Machine ClassAd

```
[  
MyType = "Machine"  
TargetType = "Job"  
Machine = "nostos.cs.wisc.edu"  
Requirements =  
(LoadAvg <= 0.300000) &&  
(KeyboardIdle > (15 * 60)) ←  
Rank = other.Department==self.Department  
Arch = "INTEL"  
OpSys = "LINUX"  
Disk = 3076076  
]
```

- classadd usados por recursos e por agentes
- dois atributos têm significado especial:
 - requirements
 - rank
- anúncios podem se referir a atributos do candidato a casamento!



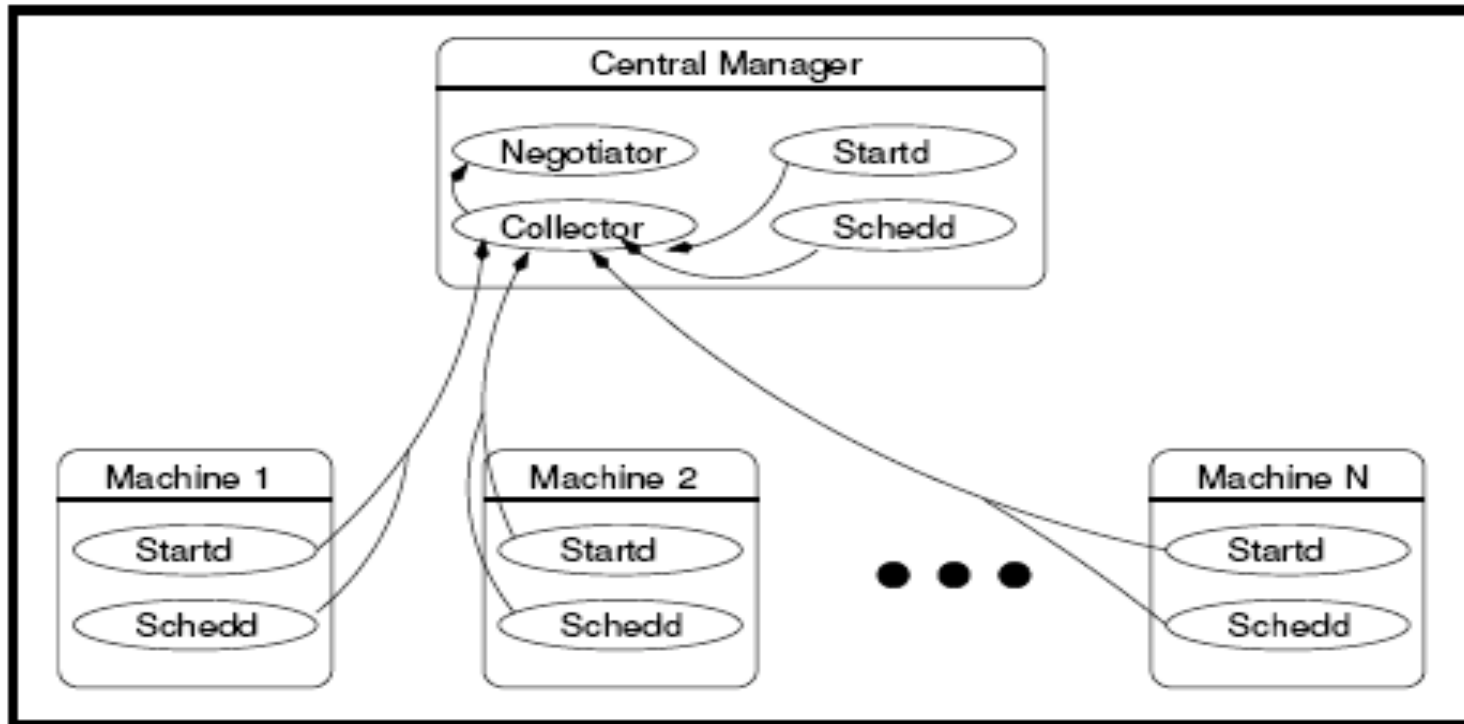
Matchmaking



- flexibilidade no aproveitamento de recursos
 - estações ociosas
 - clusters em exclusiva



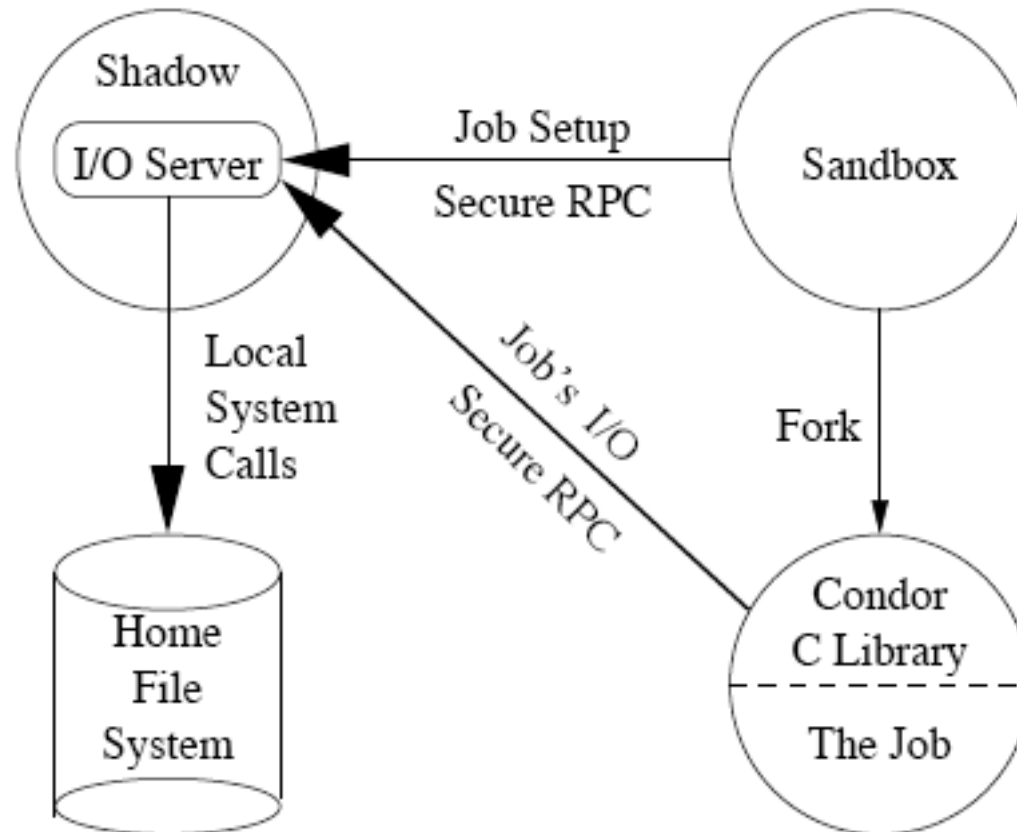
Condor: coleta de estado



- anuncios criados dinamicamente, de acordo com estado do recurso
 - startd



execução partida



- estrutura básica
 - *universos* distintos implementam variantes

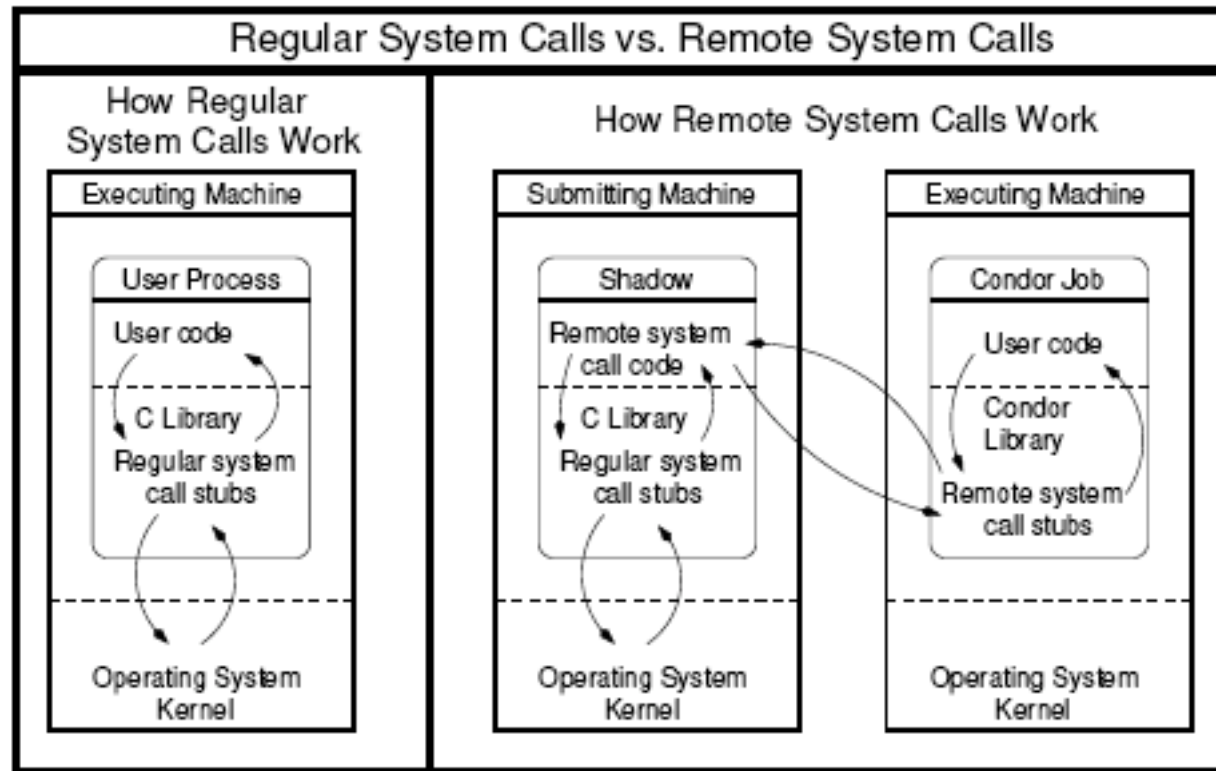


execução partida

- processo deve ser ligado com biblioteca Condor
 - wrappers de chamadas de sistema: RPC para origem
 - transformações de operações de e/s
- sandbox garante execução protegida
 - acessos a sistema são na máquina origem
 - userid novo para cada job disparado



Chamadas Remotas



- similaridade com MOSIX
 - aqui acima do kernel!



Condor - segurança

- biblioteca CEDAR permite negociação entre clientes e servidores
 - definição de algoritmos de autenticação e de proteção de dados
 - permite interação com Kerberos, GSI, ...
 - estilo SASL
- execução segura
 - sandbox



Condor - uso

- arquivos de submissão contêm infos para anúncio e para execução do job
 - comandos (linha de comando) permitem acompanhar execução

```
# Example 3: Submit lots of runs and use the
# pre-defined $(Process) macro.
universe = vanilla
executable = foo
requirements = Memory > 128  && Machine != "server-node.cluster.edu"
rank = KFlops
image_size = 180

Error    = err.$(Process)
Input    = in.$(Process)
Output   = out.$(Process)
Log      = foo.log

queue 150
```



exemplo - visão da aplicação

- sistema SAMBA
 - Alexandre Plastino. Balanceamento de Carga de Aplicações Paralelas SPMD. Tese de Doutorado. PUC-Rio, 2000.
- suporte ao desenvolvimento de aplicações SPMD com balanceamento de carga



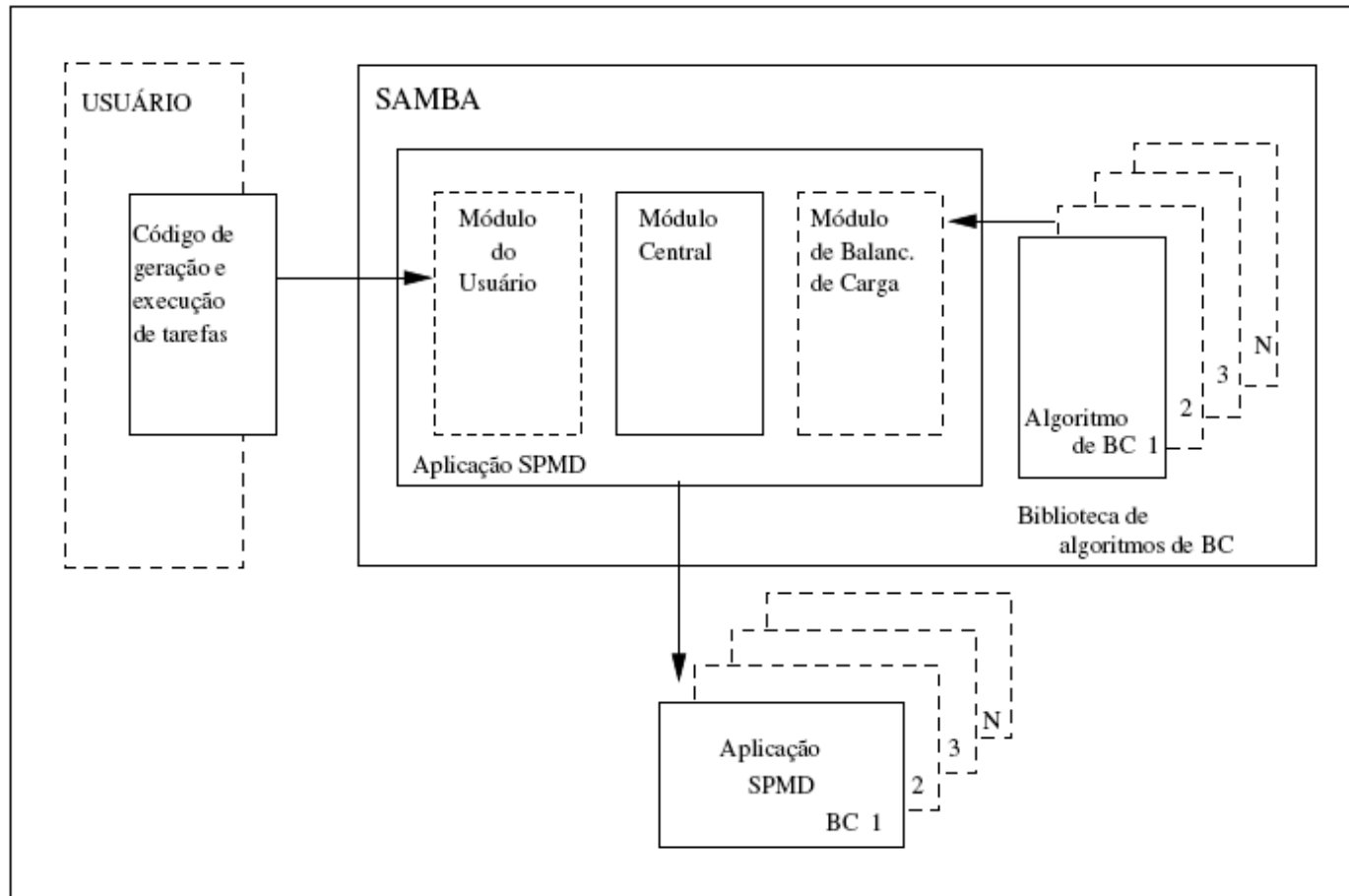
SAMBA - motivação

- muitas aplicações tem uma mesma estrutura
 - inicialização
 - execução de tarefas
 - * intercalada com balanceamento
 - coleta de resultados

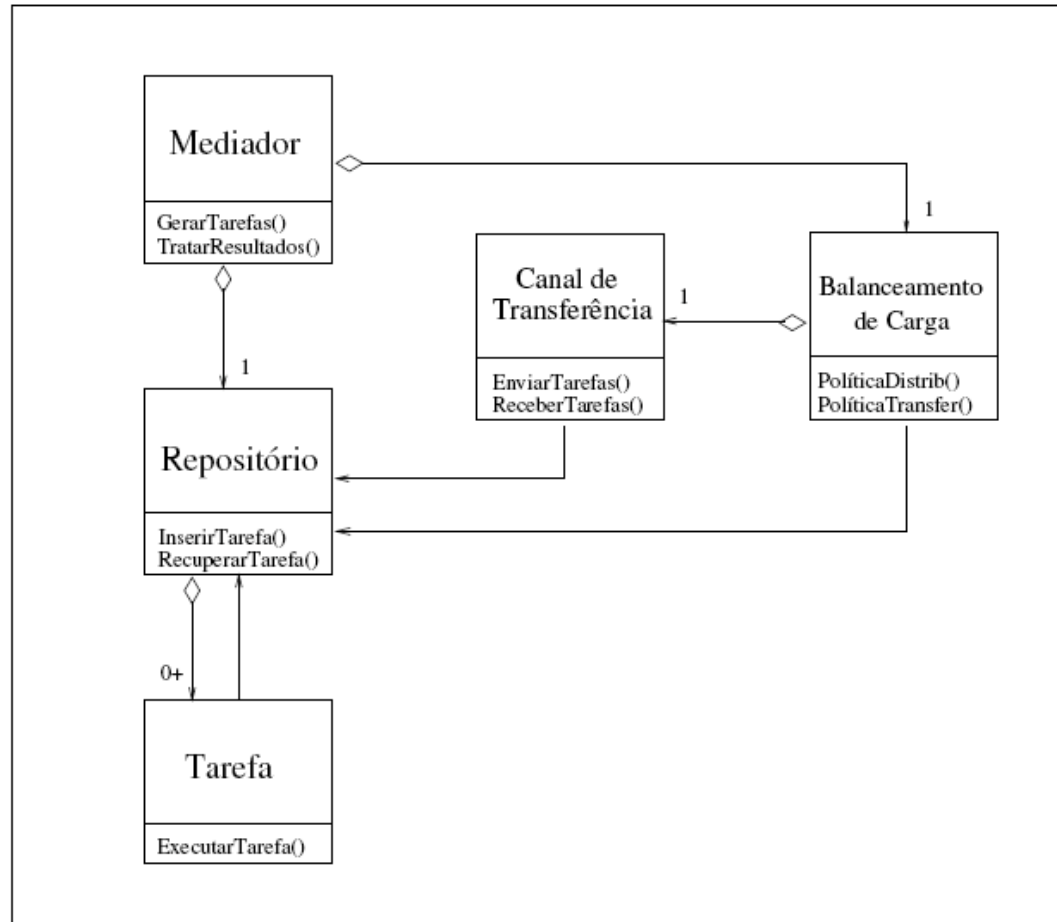
→ especialmente aplicações embarçosamente paralelas
- possibilidade de capturar essa estrutura
 - programador escrever apenas código específico de sua aplicação



SAMBA - arquitetura



SAMBA



SAMBA

- principais rotinas a serem escritas pelo programador:
 - GerarTarefas
 - » “callback” para a função InsereTarefa
 - ExecutarTarefas
 - TratarResultados
- com um pouco mais de detalhes...
 - U_inicial_mestre
 - U_inicial_escravo
 - U_executa_tarefa
 - U_final_mestre
 - U_final_escravo



exemplo: mult de matrizes

```
void U_Inicial_Mestre(argv,size)
char *argv[];
int size;
{
    int i;
    Le_Matriz(MA,argv[1],&nlina,&ncola);
    Le_Matriz(MB,argv[2],&nlinb,&ncolb);
    if (ncola!=nlinb)
        C_reporta_erro(C_ERRO,
            "\n\n Estas matrizes não podem ser multiplicadas.\n\n");
    Envia_Replica_Matriz(MB,nlinb,ncolb);
    /* Geração das tarefas. */
    for (i=0;i<nlina;i++)
        GR_Inserte_Tarefa(MA[i],sizeof(int)*ncola,i);
}
```



exemplo: mult de matrizes

```
void U_Inicial_Escravo()  
{  
    Recebe_Replica_Matriz(MB,&nlinb,&ncolb);  
}
```

```
void U_Final_Escravo()  
{  
    Envia_Resultado(MR,ncolb);  
}
```



exemplo: mult de matrizes

```
void U_Final_Mestre(size,argv)
int size;
char *argv[];
{
    Recebe_Resultados(MR,ncolb,size);
    Grava_Matriz(MR,nlina,ncolb,argv[3]);
}
```



exemplo: mult de matrizes

```
void U_Executa_Tarefa(pt_tar_usu,tam_tar_usu,id_tar_usu)
void *pt_tar_usu;
int tam_tar_usu;
int id_tar_usu;
{
    int M[MAXNCOL];
    int i,j,k;
    /* Desempacotamento da tarefa. */
    ncola=tam_tar_usu/sizeof(int);
    for(i=0;i<ncola;i++)
    {
        M[i]*=((int *)pt_tar_usu);
        pt_tar_usu = (void *)((long)pt_tar_usu + sizeof(int));
    }
    /* Execução da tarefa: multiplicação de uma linha por
        todas as colunas da segunda matriz. */
    for (j=0; j<ncolb; j++)
        for (k=0; k<ncola; k++)
            MR[id_tar_usu][j] = MR[id_tar_usu][j] + (M[k] * MB[k][j]);
    MR[id_tar_usu][MAXNCOL] = 1;
}
```



classificação de algoritmos

- utilização de índices: integradas X isoladas
- escopo: globais X locais
 - locais: particionados e por vizinhança
- exec do algoritmo: centralizada X distribuída
 - distribuídos: síncronos X assíncronos
- ativação: periódica X por evento
 - eventos: sobrecarga, subcarga
- alvo do algoritmo: individual X coletivo
- direção de transfs: receptoras X transmissoras



SAMBA

- biblioteca com 9 algoritmos de balanceamento de carga
 - estático
 - sob demanda
 - distribuído, síncrono global, coletivo, por evento, isolado
 - centralizado, global, por evento, coletivo, isolado
 - distribuído, assíncrono, global, não-cego
 - ...
- algum suporte ao desenvolvimento de outros algoritmos de BC



Referências

- Barak A., Geday S. and Wheeler R., The MOSIX Distributed Operating System, Load Balancing for UNIX. Lecture Notes in Computer Science, Vol. 672, Springer-Verlag, 1993.
- Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed Computing in Practice: The Condor Experience. *Concurrency and Computation: Practice and Experience*, 17(2-4), pages 323-356, 2005.
- A. Plastino, C. Ribeiro e N. Rodriguez. Developing SPMD Applications with Load Balancing. *Parallel Computing*, 29(6), pages 743-766, 2003.

