Aula 10 - Gerenciamento de recursos

Além de prover comunicação, recursos de acesso a rede, memória compartilhada, sistemas de arquivos distribuídos; um sistema operacional distribuído tem que poder executar um processo em um processador que se encontra em um nodo remoto, ou migrar um processo de um nodo para outro pois:

- pode ser necessário desligar o nodo
- nodo não possui algum recurso necessário para o processo
- deseja-se melhorar o tempo de execução geral do sistema

Os algoritmos de alocação de recursos (processador) devem levar em conta requisitos tais como: tempo de resposta, carga da rede, *overhead*, maximização do uso do processador.
Existem três técnicas básicas:

- atribuição de tarefas (*task assignment*): objetivo de melhorar a performance. Cada processo é visto como um conjunto de tarefas.
- balanceamento de carga (*load balancing*): distribui os processos entre os nodos de forma a tornar a carga nestes nodos similar.
- compartilhamento de carga (*load-sharing*): garantir que nodo algum fique ocioso enquanto algum processo espera para ser executado.

Características desejáveis

- sem conhecimento prévio a respeito dos processos, pois este conhecimento pode causar uma carga extra para o usuário
- dinâmico, ou seja, baseado no estado atual do sistema e não em um estado estático (fixo)
- rápido na tomada de decisão, por exemplo, algoritmo que possui uma solução próxima da ótima e é rápido para calcular, é melhor que o algoritmo que possui a solução ótima, mas é demorado para calcular
- performance equilibrada e pouco overhead no algoritmo de scheduling (quanto mais informação melhor ?)
- estável, por exemplo, que evite o problema *processor trashing*, ou seja migração constante de processos
- "escalável", tolerante a falhas, justo

Atribuição de tarefas

Cada processo é dividido em tarefas e a meta é encontrar a melhor forma de distribuir estas tarefas para um determinado processo.
Considerações:

- cada processo já está dividido em tarefas
- o tempo de execução necessário para cada tarefa e a velocidade do processador é conhecido
- o custo para executar cada tarefa em um processador é conhecido (baseado no item anterior)
- o custo de comunicação entre processos é conhecido (custo = 0, se tarefas estão no mesmo processador)

Baseado nestas considerações busca-se:

- i) minimizar os custos de IPC; ii) melhor performance por processo executado; iii) aumento no paralelismo; iv) utilização eficiente dos recursos do sistema.

Como encontrar a atribuição ótima?

Uso de um grafo, onde arcos entre tarefas representam o custo de comunicação e arcos entre tarefa e nodo representam o custo de execução no outro nodo. Solução para dois nodos. Ver exemplo (Pradeep página 353).

Balanceamento de carga

Baseado na intuição de que para melhor utilizar os recursos do sistema, a carga deve estar distribuída igualmente no sistema. Assim a idéia é ter a "mesma" carga em todos os nodos do sistema, transferindo processos de um nodo sobrecarregado para um nodo subcarregado.

Taxonomia de algoritmos para BC

- **estático**: usa somente informações sobre o comportamento médio do sistema, ignorando o estado atual do sistema. Mais simples de ser implementado mas mais limitado em potencial.
  - **determinístico**: usa inormações sobre as propriedades dos nodos e as características dos processos a serem carregados, e.g. atribuição de tarefas
  - probabilístico: usa informações de acordo com os atributos estáticos do sistema, tais como número de nodos, capacidade de processamento de cada nodo, topologia da rede, e assim por diante, de forma a formular regras de carga. Exemplo: dois processadores p1 e p2, 4 terminais t1-t4, então t1 e t2 usam p1, e t3 e t4 usam p2.

- **dinâmico**: reage de acordo com as mudanças do sistema. Melhor que os estáticos pois pode reagir a situações onde a performance do sistema é ruim. São mais complexos de serem implementados.

  - **centralizado**: responsabilidade de *scheduling* está em um único nodo. As informações sobre o estado do sistema está em um único nodo. Todas as requisições são atendidas por este nodo. Problema: *reliability* (confiabilidade): falha: k+1 replicas para tolerar k falhas. Problema, como manter as replicas consistentes?
  - **distribuído**: *scheduling* não é efetuado por um nodo, ele é distribuído entre k entidades $e_1, e_2, ..., e_k$. Cada um atuando para um conjunto de nodos. Ele pode ser completamente distribuído caso k=n.
    - **cooperativo**: trabalha em conjunto com os demais nodos. Mais complexo, mas mais estável.
    - **não-cooperativo**: cada nodo faz o *scheduling* sozinho, baseado nas informações que ele contem a respeito dos outros nodos.

Considerações em Alg. de BC

- política de estimativa de carga: como medir a carga de um determinado nodo no sistema? Qual o método para medir a carga de um nodo? Exemplos: i) número total de processos no nodo no momento da estimativa da carga; ii) demanda de recursos destes processos; iii) arquitetura e velocidade do processador do nodo; iv) soma daquilo a ser usado pelos processos naquele nodo (*memoryless*-mesmo tempo de processamento para todos os processos, independente do tempo utilizado até o momento; ou *pastrepeats*-o resto é igual aquilo já utilizado).

- política de transferência de processos: como transferir alguns processos de um nodo sobrecarregado para nodos que não estão sobrecarregados? Como decidir se um nodo está sobrecarregado? Duas formas: usando um limite; ou usando três níveis (sobrecarregado, normal, subcarregado).

- política de seleção de um novo nodo: Uma vez determinado que um processo deve ser transferido, deve-se decidir onde este processo irá ser executado. Existem 4 formas básicas:

  - limite: destino é selecionado aleatoriamente, se ele não estiver sobrecarregado o processo é enviado apra o nodo e executado lá, mesmo que o nodo esteja sobrecarregado quando o processo chegar ao nodo. Se não encontrar um nodo, então nodo original deve executar processo.

  - mais leve: L nodos são selecionados aleatoriamente, o menos carregado é selecionado para carga (caso ele não esteja sobrecarregado). Se nenhum nodo pode ser escolhido, então o processo é executado no nodo original. Um forma de melhorar este algoritmo seria parar a pesquisa caso carga zero seja encontrada.
  - lance: cada nodo possui dois papéis em relação ao processo de lances: i) comprador: nodo que precisa um local para executar um processo; ii) vendedor: nodo que pode aceitar processos. Cada comprador envia um *broadcast* para todos nodos do sistema. Os vendedores fazem um "lance" (uma mensagem com informações do nodo, e.g. tamanho de memória, disponibilidade de recursos, ...). Quando o comprador recebe a resposta, ele aceita a melhor oferta (lance).
  - pares: dois nodos que diferem em carga (um sobrecarregado e outro subcarregado) forma um par e balanceamento de carga é efetuado entre estes dois nodos temporariamente.

- política de troca de informações: quando enviar informações sobre o estado do sistema entre os nodos? i) broadcast periódico; ii) broadcast quando o estado muda; iii) broadcast quando é necessário; iv) troca de informações através de polling.
- política de prioridade: como fazer o *scheduling* entre os processos?
  - egoísta: processos locais tem maior prioridade que processos remotos (pior tempo de resposta)
  - altruísta: processos remotos tem maior prioridade que processos locais (melhor)
  - intermediário: depende do número de processos locais e remotos. Se existem mais processos locais, então processos locais tem maior prioridade.
- política de migração: i) controlada: fixa o número de vezes que um processo pode migrar; ii) não-controlada: processo chegando em um nodo é tratado como um processo local, desta forma pode migrar novamente.

Compartilhamento de Carga

É necessário e suficiente prevenir que nodos fiquem ociosos e outros tenham mais que 2 processos.