UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de São José do Rio Preto

Wellington Francisco da Silva

Algoritmo *Particle Swarm* para Escalonamento de Máquinas Virtuais em Computação em Nuvem

Orientadora: Profa. Dra. Renata Spolon Lobato

São José do Rio Preto
2018

Wellington Francisco da Silva

Algoritmo *Particle Swarm* para Escalonamento de Máquinas Virtuais em Computação em Nuvem

.

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista “Júlio de Mesquita Filho”, Câmpus de São José do Rio Preto.

São José do Rio Preto
2018

Silva, Wellington Francisco da.
Algoritmo Particle Swarm para escalonamento de máquinas virtuais em computação em nuvem / Wellington Francisco da Silva. -- São José do Rio Preto, 2018
83f. : il., grafs., tabs.

Orientador: Renata Spolon Lobato
Dissertação (mestrado) – Universidade Estadual Paulista (UNESP), Instituto de Biociências, Letras e Ciências Exatas, São José do Rio Preto

1. Ciência da computação. 2. Computação em nuvem. 3. Memória virtual (Computação) I. Título.

CDU – 681.32

Ficha catalográfica elaborada pela Biblioteca do IBILCE
UNESP - Câmpus de São José do Rio Preto

Wellington Francisco da Silva

Algoritmo *Particle Swarm* para Escalonamento de Máquinas Virtuais em Computação em Nuvem

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista “Júlio de Mesquita Filho”, Câmpus de São José do Rio Preto.

Comissão Examinadora

Profª. Drª. Renata Spolon Lobato
UNESP – Câmpus de São José do Rio Preto
Orientadora

Prof. Dr. Rodrigo Guido
UNESP – Câmpus de São José do Rio Preto

Prof. Dr. Henrique Dezani
FATEC – São José do Rio Preto

São José do Rio Preto
04 de dezembro de 2018

AGRADECIMENTOS

À minha esposa e filha.

RESUMO

A demanda computacional dos últimos anos fez um novo paradigma computacional tornar-se extremamente necessário para suprir a demanda por recursos. A computação em nuvem tem sido muito usada e é realidade em todos setores que demandam uso computacional aliado com segurança e com facilidade de gerenciamento.

Data Centers gigantescos foram criados para atender uma demanda cada vez maior. Processamento, memória e armazenamento são entregues a clientes finais que não tem a preocupação com energia, resfriamento, hardware, software, licenças e gerenciamento, pagando apenas pelo que realmente necessita.

Considerando que o usuário solicita recursos para executar uma determinada tarefa, faz-se necessária a criação de mecanismos eficientes de alocação de recursos e métricas de cobrança justas.

Neste trabalho é feita uma revisão de conceitos de computação em nuvem, virtualização e escalonamento de recursos. São analisados alguns algoritmos de escalonamento. Utiliza o algoritmo *particle swarm* como base para escalonar máquinas virtuais na classe de infraestrutura como serviço(IaaS). Busca o ambiente que atenda a necessidade de recursos solicitados e o QoS (qualidade de serviço) contratado.

Por fim é implementado o algoritmo *particle swarm* para fazer análise da melhor configuração de parâmetros para atender a demanda de alocação de máquina virtual em computação em nuvem. É considerado para o cálculo a quantidade de *CPU*, memória e disco.

Os resultados mostraram que o algoritmo é eficiente para ser utilizado para escalonar máquinas virtuais em computação em nuvem.

Palavras chaves – Computação em nuvem, virtualização, escalonamento de recursos.

ABSTRACT

The computational demand of the last years has made a new computational paradigm become extremely necessary to supply the demand for resources. Cloud computing has been widely used and is a reality in all sectors that demand computational use allied with security and with ease of management.

Gigantic data centers were created to meet ever-increasing demand. Processing, memory, and storage are delivered to end customers who do not have the energy, cooling, hardware, software, licensing, and management concerns, paying only for what they really need.

Considering that the user requests resources to perform a certain task, it is necessary to create efficient mechanisms of allocation of resources and fair collection metrics.

In this work a review of concepts of cloud computing, virtualization and scheduling of resources is made. Some scaling and collection algorithms are analyzed. It uses the particle swarm algorithm as the basis for staging virtual machines in the infrastructure class as a service (IaaS). It seeks the environment that meets the need for requested resources and contracted QoS.

Finally, the particle swarm algorithm is implemented to make analysis of the best parameter configuration to meet the demand for virtual machine allocation in cloud computing. The amount of CPU, memory and disk is considered for calculation.

The results showed that the algorithm is efficient to be used to stagger virtual machines in cloud computing.

Key Words - Cloud computing, virtualization, resource scalability.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE GRÁFICOS

LISTA DE EQUAÇÕES

LISTA DE SIGLAS

API	Application Programming Interface
CPU	Central Processing Unit
IaaS	Infrastructure as a service
PaaS	Plataform as a Service
QoS	Quality of service
SaaS	Software as a Service
SLA	Service leve agreement
VM	Virtual Machine

Sumário

Capítulo 1 Introdução

A revolução tecnológica que a humanidade tem experimentado fez surgir uma crescente demanda por recursos computacionais. Milhares de dispositivos conectados, redes sociais, comunicação instantânea, exploração espacial, o mundo hoje é conectado e para manter tudo isto funcionando são necessários, além de uma grande rede computacional, *Data Centers* gigantescos com enorme capacidade de processamento e armazenamento.

Um problema encontrado nesta expansão é como gerenciar tudo isso de forma prática e com segurança, garantindo que os serviços sempre estejam disponíveis e com capacidade de resposta adequada. Há também a demanda energética que estes grandes *data centers* necessitam, o que é uma preocupação mundial visto a escassez de recursos que pode-se enfrentar em breve.

Várias soluções têm sido propostas e entre elas há a computação em nuvem, objeto deste estudo (HE;HE, 2011). A computação em nuvem consiste em uma nova forma de gerenciamento e disponibilização de recursos. Otimização de *hardware*, redução do custo de manutenção e acessibilidade, no qual o cliente não precisa se preocupar com *upgrade* de *hardware* e *software* (BHADAURIA; CHAKI, 2011).

De acordo com o NIST - (*National Institute of Standards and Technology*) (Mell e Grance, 2011), a computação em nuvem é como um modelo que fornece conveniência e acesso sob-demanda para recursos compartilhados (redes, servidores e serviços) que podem ser rapidamente entregues quase sem nenhum esforço de gestão por parte dos usuários.

A computação em nuvem mascara a complexidade de gerenciamento para o usuário final. Ela implementa recursos e os disponibiliza aos usuários finais, por sua vez, este usuário não sabe necessariamente onde seus dados estão armazenados ou onde seus sistemas estão sendo processados. Por isso surgiu o termo Computação em Nuvem (LEE, 2010, WANG et al., 2008).

Os serviços executados em nuvem vão desde simples aplicativos a algoritmos de previsão do tempo e processamento de imagens espaciais. Estes

serviços são executados em grandes *data centers,* geralmente distribuindo os dados geograficamente para garantir a disponibilidade e segurança dos mesmos. A virtualização é uma forma de otimizar a utilização de recursos computacionais trazendo maior escalabilidade no fornecimento de recursos (TIAN, 2015).

1.1 Objetivos

Neste trabalho é apresentada uma revisão bibliográfica sobre computação em nuvem, suas implicações, benefícios e dificuldades.

Também é discutido o escalonamento de recursos na classe de Infraestrutura como serviço e implementado o algoritmo *particle swarm* para fazer o escalonamento de máquinas virtuais para o *data center* mais adequado para atender demanda solicitada.

São demonstrados estudos de caso relevantes de algoritmos de outros trabalhos para serem utilizados a título de comparação e elaborada uma tabela de classificação com as particularidades de cada um.

O algoritmo utilizado como base para a criação do escalonador é o *particle swarm,* que será explicado na seção 3.3.4.

Este trabalho está organizado da seguinte forma:

• No capítulo 2 é fundamentada a computação em nuvem com seus principais aspectos, como classes(*software* como serviço, plataforma como serviço e infraestrutura como serviço), modelos de computação em nuvem(pública, privada, híbrida ou comunitária), *Intercloud,* alta disponibilidade, segurança na computação em nuvem. É apresentada a virtualização e como ela é importante na computação em nuvem.

• Capítulo 3 está organizado para esclarecer sobre as métricas de utilização e cobrança e também os algoritmos de escalonamento.

• No capítulo 4 são descritos os detalhes sobre esta pesquisa e apresentada a metodologia.

• No Capítulo 5 são apresentados os resultados.

• E por fim, no capítulo 6 a conclusão.

Capítulo 2 Computação em Nuvem

De acordo com o NIST - *(National Institute of Standards and Technology) (Mell e Grance, 2011),* a Computação em Nuvem é um modelo que permite ubiquidade, conveniência e acesso sob-demanda para a utilização de recursos computacionais compartilhados que podem ser rapidamente entregues com um esforço mínimo de gestão por parte do solicitante.

A meta é entregar recursos de acordo com a necessidade e que isto seja feito de forma transparente para quem esteja solicitando o recurso. Utilizando conceitos, como virtualização e emulação, monta uma nova arquitetura, os reuni e disponibiliza recursos sob demanda.

Segundo Buyya et. al. (2011) a virtualização de hardware permite criar múltiplas máquinas virtuais sobre uma máquina física. Assim, a computação em nuvem utiliza a virtualização para criar um ambiente e disponibilizar serviços de acordo com os recursos disponíveis. Essas instâncias de máquinas virtuais são alocadas de acordo com as máquinas físicas que compõem o conjunto de *data centers* da nuvem.

2.1 Definição de Computação em Nuvem

A computação em nuvem é definida como o compartilhamento de recursos computacionais através de computadores interligados em rede. O uso de processamento, memória e armazenamento é feito sob demanda de acordo com a necessidade. Como este compartilhamento é feito através de uma rede (que pode ser a internet), a distribuição de recursos é feita de acordo com a necessidade e é utilizado como um serviço, neste caso quem utiliza não sabe necessariamente onde seus dados estão sendo processados. O que interessa é a entrega final do que foi solicitado, daí o termo “computação em nuvem”.

Ainda de acordo com o NIST, é possível fazer quatro divisões na definição de computação em nuvem:

• Autoatendimento: é possível provisionar recursos de computação, sem a necessidade de interação humana com cada prestador de serviço.

• Acesso à rede: O acesso a recursos disponibilizados sobre a rede é realizado através de quaisquer tipos de plataforma, como por exemplo, telefones celulares, *tablet*, *notebooks*, e estações de trabalho.

• Agrupamento: os recursos atendem múltiplos consumidores com diferentes recursos físicos e virtuais atribuídos dinamicamente.

• Elasticidade: provisionamento e liberação elástica de recursos, para se ajustar a necessidade, crescente ou decrescente.

A computação em nuvem é desenvolvida com base em várias progressões recentes em virtualização, computação distribuída, computação em grade, computação na web e computação autônoma. Também por definição, um ambiente de computação em nuvem possui um grande conjunto de usuários utilizando facilidade de acesso.

Recursos virtualizados (como hardware, plataformas de desenvolvimento e/ou serviços), podem ser reconfigurados dinamicamente para ajustar a uma carga variável permitindo também uma utilização de recursos adequados. Agendar os trabalhos é um desafio para a computação em nuvem, visto que são milhões de usuários requisitando recursos. O algoritmo de agendamento baseado em regras (por exemplo, exaustivo e algoritmo de agendamento determinista) são simples e fáceis de implementar, portanto, eles são amplamente usados atualmente em sistemas de computação em nuvem, mas em problemas de agendamento em larga escala o resultado deste método de agendamento geralmente está longe de ser ótimo e não é apropriado (ARDABILI, 2016).

A computação em nuvem elimina o gasto com a compra de *hardware* e *software* e instalação e execução de *data centers* locais. Reduz custo com eletricidade que demanda disponibilidade permanente para energia e resfriamento, e também custo com especialistas para gerenciamento da infraestrutura.

A maior parte dos serviços de computação em nuvem é fornecida por autoserviço e sob demanda, desta forma até grandes quantidades de recursos

de computação podem ser provisionados em minutos, normalmente com apenas alguns cliques, fornecendo às empresas muita flexibilidade e aliviando a pressão do planejamento de capacidade.

Por conta de sua capacidade de dimensionamento elástico, é possível fornecer quantidade correta de recursos, por exemplo, mais ou menos energia de computação, armazenamento e largura de banda, quando necessário e no local geográfico correto.

É possível alcançar níveis altos de desempenho pois os *data centers* são atualizados regularmente com a mais recente geração de *hardware* de computação rápido e eficiente. Isso oferece diversos benefícios em um único *data center* corporativo, incluindo latência de rede reduzida para aplicativos e mais economia de escalonamento.

Um ponto importante é que ao invés de investir substancialmente em *data centers* e servidores antes de saber como serão utilizados, pode-se pagar apenas quando consumir recursos de computação, e pagar apenas pela quantidade consumida. É possível alcançar um custo variável mais baixo do que seria possível normalmente.

2.2 Classes de Computação em Nuvem

É possível dividir a computação em nuvem em várias classes, mas as principais são três e serão explanadas nesta seção. Em Buyya (2011) é possível observar esta classificação, ela é dividida em três classes de serviços de acordo com o modelo de serviços oferecidos pelos provedores: Infraestrutura como um Serviço (IaaS), Software como um Serviço (SaaS) e Plataforma como um Serviço (Paas).

Na Figura 1 são ilustradas as camadas apresentadas que formam a base da computação em nuvem. É possível observar a distribuição de serviços através das camadas com exemplos atualmente oferecidos pelas principais empresas provedoras desta modalidade.

Figura 1 - Computação em nuvem. Fonte: Buya et al., (2011).

2.2.1 Software como serviço

No Software como um Serviço (SaaS – *Software as a Service*), o software que antes era instalado no computador do usuário, agora é oferecido como um serviço. O que o usuário necessita é apenas acesso a internet para utilizar o *software*, desta forma isto pode ser feito de qualquer lugar e na maioria dos casos, de qualquer dispositivo. Pode-se citar como exemplo o *Office* 365 da Microsoft ou Google *Docs* da Google.

2.2.2 Plataforma como serviço

A Plataforma como um Serviço (PaaS – *Plataform as a Service*) é fornecida como um serviço, por exemplo, plataformas para desenvolvimento, como o Google Apps Engine e Microsoft Windows Azure. Pode ser usada para criar aplicativos *web* (SaaS), fornece um ambiente integrado de desenvolvimento e ambientes de teste (RIMAL et al., 2009).

2.2.3 Infraestrutura com serviço

A categoria de Infraestrutura como um serviço (IaaS - *Infrastructure as a service*) é a infraestrutura computacional que pode ser fornecida como um serviço através da tecnologia de virtualização. Geralmente fornece acesso a estrutura dedicada (virtual ou hardware) de recursos de rede e computacional e até mesmo armazenamento de dados. É muito semelhante a ter uma estrutura própria de recursos em d*ata centers* próprios, com a diferença de não haver preocupação com gerenciamento físico da estrutura com todas as complexidades envolvidas.

Na Figura 2 é detalhada cada camada da computação em nuvem.

Figura 2: Arquitetura detalhada da computação em nuvem. Fonte: Adaptado de Zhang et al. (2010).

É possível observar que o gerenciamento de recursos em baixo nível só acontece na Infraestrutura como serviço, neste tipo é contratada uma estrutura parecida com *data centers* locais e faz-se necessário que recursos como CPU, memória e disco sejam alocados de acordo com a necessidade da tarefa.

Já na plataforma como serviço toda estrutura já está montada e as tarefas são executadas em sistemas prontos, como por exemplo, uma plataforma *online* de desenvolvimento de *softwares.*

No *software* como serviço, o usuário final consome recursos de uma aplicação específica e não tem a preocupação com a plataforma que este *software* está executando e nem com recursos de processamento, memória ou disco.

Considerando as restrições de acesso, é possível identificar quatro modelos de implementação de nuvens que serão mostrados nos próximos itens (CAROLAN et al., 2009) (MELL e GRANCE, 2011).

Os modelos não representam necessariamente a localização física da nuvem, e sim apenas como será utilizada. Mesmo em casos de nuvem privada os clientes não precisam saber onde seus dados estão sendo processados ou armazenados.

2.3 Modelos de computação em nuvem

Sasikala (2012) divide os modelos de nuvem de acordo com o grupo que irá utilizar os serviços, podendo ser de quatro tipos: nuvem pública, privada, comunitária e híbrida, a Figura 3 ilustra o relacionamento entre os modelos de computação em nuvem.

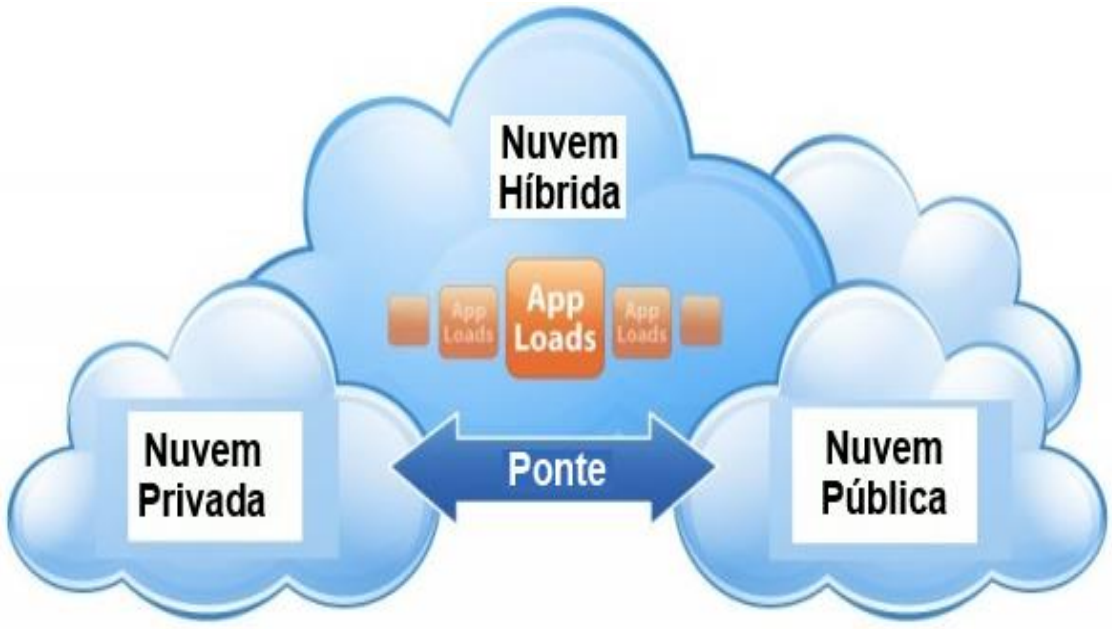

Figura 3 - Modelos da computação em nuvem. (THANGARAJ, 2012)

2.3.1 Nuvens Públicas

Nuvens públicas são executadas por terceiros. Fornecem serviços e infraestrutura para os consumidores. Este modelo é ideal para empresas que precisam de serviços flexíveis e temporários para executar uma tarefa, também para garantir a redução de risco e custo. Os serviços da nuvem podem ser gratuitos ou vendidos pelo provedor da nuvem.

Neste modelo toda responsabilidade de gerenciar, armazenar, hospedar e prover segurança é responsabilidade do fornecedor. Embora o contratante do serviço tenha total controle sobre seus dados, não é possível saber onde esses dados estão armazenados e onde o serviço está executando. Porém é possível obter boa economia visto que os serviços são pagos sob demanda.

Uma nuvem pública pode ter mais de um provedor de serviços como também mais de um usuário usando os serviços. De acordo com as requisições que chegam, o gerenciador de nuvem aloca recurso de acordo com a disponibilidade de algum provedor desta nuvem. Esta alocação pode ser baseada em algumas métricas, como por exemplo, distância física e capacidade do provedor em entregar o serviço de acordo com o SLA (acordo de nível de serviço) contratado.

2.3.2 Nuvens Privadas

No modelo de nuvem privada a gestão pode ser personalizada para as necessidades do cliente. Pode ser alugada ou de propriedade do cliente.

O uso de nuvens privadas é restrito a uma companhia ou determinado grupo, não havendo compartilhamento de recursos. A própria companhia determina as regras de implementação e uso de sua nuvem, que pode ser em estrutura própria ou até mesmo contratada de outro provedor, só que com a diferença de ser exclusiva para a empresa que contratou o serviço.

Empresas que desejam manter total controle dos seus dados para as quais seus serviços estão hospedados e para isso preferem o uso de infraestrutura própria, tem na nuvem privada uma ótima solução para otimizar o uso de recursos de seu *data center*.

Porém, o custo de implementação e gerenciamento de uma nuvem privada pode ser um empecilho para empresas de pequeno e médio porte.

2.3.3 Nuvens Híbridas

Uma alternativa para ajudar a resolver o problema do alto custo para manter nuvens privadas é o uso das nuvens híbridas.

Nuvem híbrida é a combinação de nuvens públicas com nuvens privadas e essa combinação visa melhorar a escalabilidade sob demanda. Neste modelo, quando uma nuvem privada tem uma carga de trabalho alta, ela temporariamente compartilha o uso de recursos públicos para garantir o desempenho.

É possível determinar quais serviços serão executados na nuvem pública e quais serão executados na nuvem privada, trazendo maior flexibilidade à estrutura e também reduzindo o custo de implantação e gerenciamento.

Também há a infraestrutura de nuvem comunitária que é provisionada para o uso de uma comunidade específica de consumidores que tem objetivos comuns, podendo ser controlada, gerenciada e operada por uma ou mais organizações pertencentes a essa comunidade.

2.4 *Intercloud*

O modelo de computação em nuvem, assim como outros modelos, possui riscos. Há o risco de rejeição de um aplicativo na nuvem, as regras de negócio não serem atendidas com os mesmos níveis de serviço requeridos para o serviço. Também quando o serviço de nuvem depende necessariamente

da internet, pode haver problema com relação a disponibilidade do serviço de internet em determinada região, o que causaria a indisponibilidade do serviço. É preciso também que haja uma maior regulamentação, pois cada provedor pode ter regras diferentes com níveis diferentes de transparência

Faz-se necessário que os benefícios e riscos devam ser analisados individualmente para cada projeto e serviço que deva ser migrado para nuvem.

Um grande problema enfrentado é a dificuldade em prover qualidade na entrega do serviço, principalmente em provedores de nuvem com apenas um *Data Center*. A quantidade de recursos é limitada e pode chegar uma demanda com uma quantidade alta de usuários que este provedor sozinho não terá capacidade de atender (ASSUNÇÃO et al., 2010).

Neste contexto surge a *Intercloud*, com o propósito de melhorar QoS (*Quality of Service*) em balanceamento de carga, capacidade de resposta e custo menor. Para isto utiliza a entrega de serviços sob demanda e técnica de virtualização.

Consiste na união de provedores de nuvem e a disponibilização de recursos de forma unificada, transparente para o usuário e disponível de acordo com a necessidade. Desta forma caso chegue uma solicitação de recursos que um provedor não tenha capacidade de atender, esta demanda será direcionada automaticamente para outro provedor que tenha condições de atender a demanda, formando assim uma entidade colaborativa (SOTIRIADIS et al., 2011).

Segundo (Buyya et al., 2010), esta forma de trabalho usando plataformas abertas para unificação facilita bastante a entrega de QoS (Qualidade de Serviço) para provedores de nuvem.

Há um serviço que centraliza e gerencia a demanda tanto de produtores quanto de consumidores. Os coordenadores de nuvem são responsáveis por informar o quanto de recursos tem disponíveis e a qual custo. Com esta informação o serviço principal gerencia as cargas de trabalho que chegam e as distribui, remove ou remaneja. Desta forma o cliente tem a sua disposição, várias opções de escolha de acordo com a demanda de trabalho atual,

podendo considerar custos e capacidade de execução que necessita para cada atividade.

Usando as funções de negociação do centralizador, todo cliente pode estabelecer contratos dinamicamente com os coordenadores de nuvem.

Na Figura 4 é ilustrado o funcionamento da *Intercloud*.

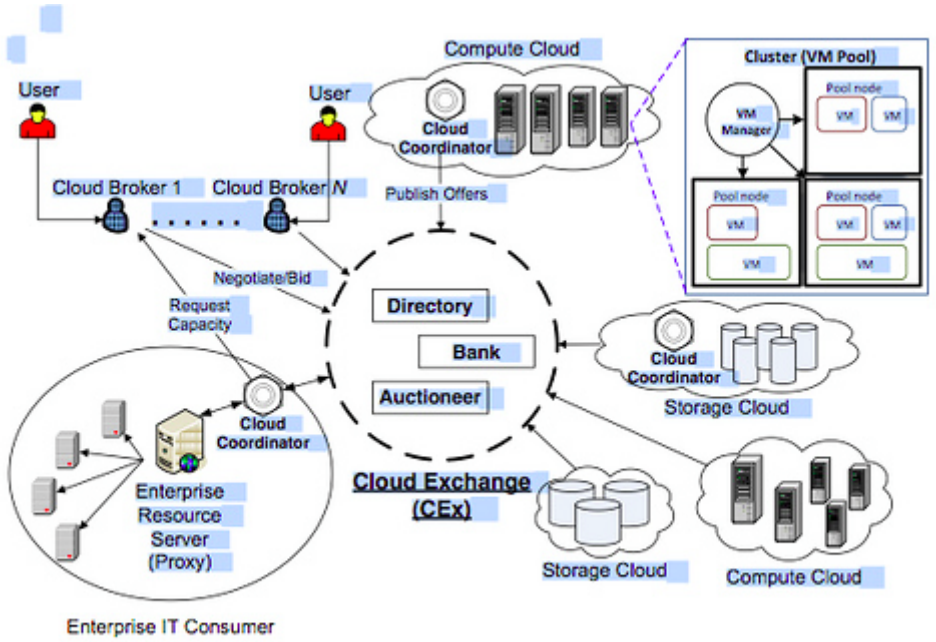

Figura 4 – Intercloud. Fonte: Buyya (2010)

O coordenador é responsável pelo gerenciamento de sua nuvem, considerando protocolos estabelecidos pelo mercado e pela corporação que disponibiliza os serviços. Ele é o responsável por disponibilizar recursos ao centralizador e também por fazer internamente alocação dos recursos requisitados.

Ainda segundo Buyya (2010), o coordenador é responsável pelo agendamento e alocação das máquinas virtuais para os nós da nuvem, baseado no QoS do usuário e no gerenciamento de energia. Ao receber uma aplicação de usuário, o agendador busca o *Application Composition Engine* sobre disponibilidade de software e serviços de infraestrutura de hardware necessários para satisfazer o pedido localmente, solicita ao componente *Sensor* que envie comentários sobre a energia local dos nós, questiona o

Market and Policy Engine (Figura 5) sobre a responsabilidade do pedido enviado. Caso o usuário possua créditos disponíveis baseado nas restrições de QoS especificadas, o SLA é estabelecido. No caso de tudo estar correto, o aplicativo é hospedado localmente e é periodicamente monitorado até que ele termine a execução.

Os recursos do centro de dados podem fornecer diferentes níveis de desempenho aos seus clientes, desta forma a seleção de recursos para atingir QoS desempenha um papel importante na nuvem. Além disso, os aplicativos da nuvem podem apresentar cargas de trabalho variadas. Torna-se essencial realizar um estudo de serviços de nuvem e suas cargas de trabalho para identificar comportamentos comuns, padrões e explorar a previsão de carga e abordagens que potencialmente podem levar a agendamento e alocação mais eficientes.

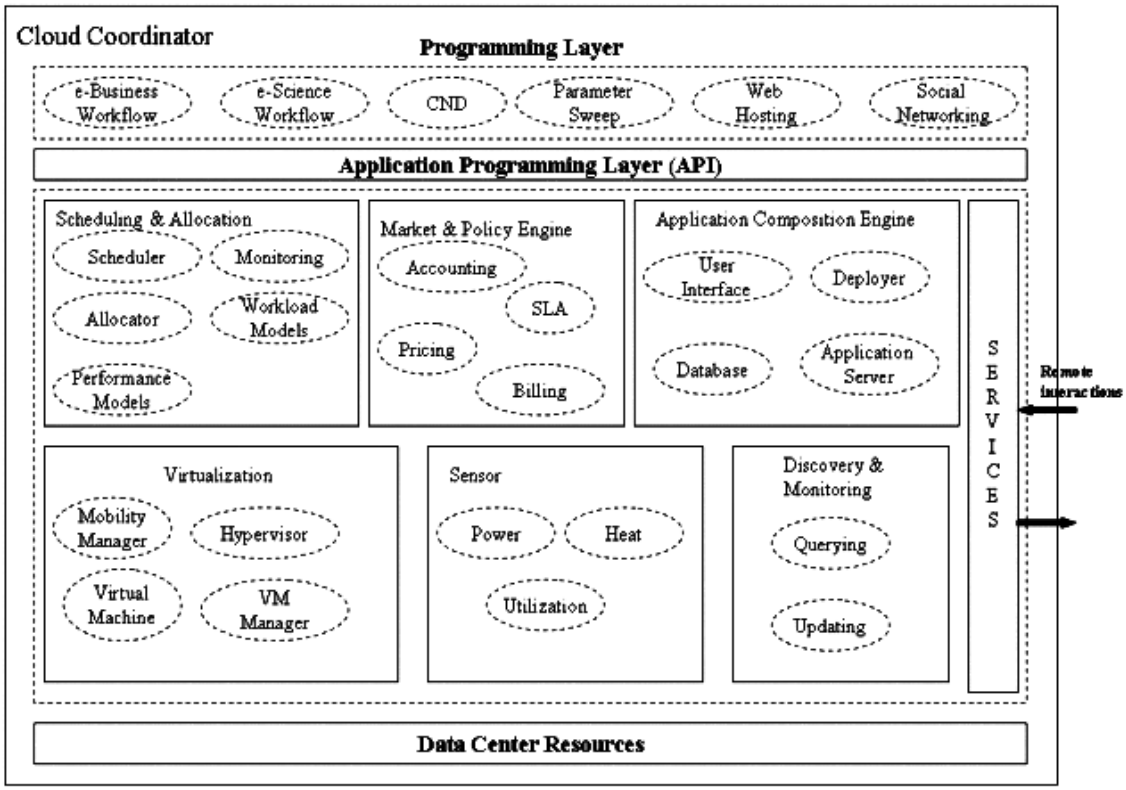

figura 5 - Coordenador de nuvem. Fonte: Buyya (2010)

O *Market and Policy Engine* (Figura 5) é o módulo que armazena os termos do serviço, SLA e condições que estão sendo suportadas pela nuvem para cada respectivo *cloud broker.* Com base nestes termos e condições, o

módulo de preços pode determinar como os pedidos de serviço são cobrados com base no fornecimento disponível. O módulo de contabilidade armazena a informação de uso real de recursos por solicitações para que o uso total do custo de cada usuário possa ser calculado.

O *Cloud Broker* que atua em nome dos usuários identifica o serviço adequado da nuvem através do centralizador de nuvem e negocia com os coordenadores da nuvem por uma alocação de recursos que atende às necessidades de QoS dos usuários.

2.5 Segurança na Computação e nuvem

Segundo Sabahi (2011), a segurança na computação em nuvem é item fundamental a ser considerado em qualquer projeto de nuvem. Existem muitos pontos de vulnerabilidades existentes na nuvem, e o problema não é somente encontrá-los, e sim aplicar resposta adequada. Os gestores de nuvem devem possuir mecanismos de armazenamento especializado, dirigido por um coordenador de transações distribuídas, que suporte alta disponibilidade. Para atingir a flexibilidade, escalabilidade e eficiência de utilização dos recursos disponíveis, os provedores de nuvem devem enfrentar grandes desafios na área da adaptabilidade, descarga de trabalho e adaptação. A segurança na computação em nuvem não difere muito de estrutura tradicional e deve ser feito em camadas. A utilização de equipamentos, como por exemplo, *firewall* é fundamental para evitar ataques comuns como *Dos, XSS, SQL Injection* e ataques de força bruta.

A *Cloud Security Alliance* (CSA) é uma organização sem fins lucrativos que tem como objetivo promover a utilização das melhores práticas para computação em nuvem. A *Cloud Security Alliance* é formada por profissionais da indústria, empresas, associações e outras partes interessadas. (CSA, 2012). Segundo relatos da CSA (2012), os treze domínios de segurança que uma nuvem deve englobar são:

- Arquitetura de *Framework* da Computação em Nuvem;
- Governança e Gestão de Riscos Corporativos;

• Descoberta Legal e Eletrônicos;

• Observância e Auditoria;

• Informação de Gestão de Ciclo de Vida;

• Portabilidade e Interoperabilidade;

• Segurança Tradicional, Continuidade de Negócios, e Recuperação de Desastres;

• Operações de centros de dados;

• Resposta a incidentes, notificação e remediação;

• Segurança da aplicação;

• Criptografia e gerenciamento de chaves;

• Gerenciamento de identidade e acesso;

• Virtualização.

Esses itens ajudam a garantir a proteção de dados contra problemas de segurança.

2.5.1 Problemas de segurança na Computação em Nuvem

CSA divulgou uma lista com os principais problemas de segurança na computação em nuvem (CSA, 2012).

• Perda de dados ou vazamento: Aplicativos podem deixar dados vazarem por vários motivos como descontrole de APIs, geração de chaves ou armazenamento irregular. Em alguns casos não há políticas de destruição de dados, não tendo o usuário garantias que realmente seus dados foram excluídos;

• Vulnerabilidade de tecnologias compartilhadas: Uma configuração errada pode afetar vários servidores e máquinas virtuais que compartilham esta informação. Devem existir acordos de nível de serviço (SLAs) para garantir gerenciamento de atualizações e melhores práticas para manutenção da rede e configuração.

• Internos maliciosos: Cada provedor tem seus próprios níveis de segurança sobre o acesso aos centros de dados. São necessários diferentes níveis de controle e acesso aos dados por parte da equipe.

• Desvio de tráfego, contas e serviços: É necessário especial atenção à autenticação, pois se a mesma for feita de forma insegura pode colocar em risco o acesso a todos esses itens. Este acesso pode ser com acesso à conta do cliente, neste caso, um cliente tem todo conteúdo de sua máquina virtual exposto ao invasor. Tem também o acesso ao administrador da nuvem, que neste cenário, o invasor tem poder sobre todas as máquinas virtuais de todos os clientes, o que será um problema bem maior.

• Interfaces de programação de aplicativos (APIs) inseguras: APIs com regras de segurança fracas podem proporcionar que usuários mal intencionados utilizem de seus serviços para invadirem contas.

• Abuso e uso não autorizado da computação em nuvem: é quando os serviços hospedados na nuvem sofrem acesso por pessoas não autorizadas com fins mal intencionados, como quebra de senhas ou outras ameaças para negócios.

• Perfil de risco desconhecido: se por um lado a transparência facilita algumas coisas para o desenvolvedor que utiliza a nuvem, por outro lado essa transparência faz o contratante ver apenas uma interface, não ficando claro as configurações e quais regras de segurança são aplicadas.

2.6 Virtualização na Computação em Nuvem

A virtualização possibilita a execução de mais de uma máquina em apenas um hardware. Utilizando os recursos da máquina física e fazendo o gerenciamento de forma otimizada é possível aproveitar melhor estes recursos e assim reduzir o custo com hardware e gerenciamento.

A interface entre a máquina física e as máquinas virtuais é feito pelos *hipervisores* que são responsáveis pela comunicação das máquinas virtuais com o hardware.

O *hipervisor* (Figura 6) é responsável por construir as interfaces virtuais, baseadas nos recursos físicos disponíveis. Também garante que cada máquina virtual seja independente de forma que uma não interfira no funcionamento de outra e não dependa da liberação de recursos de outra para realizar suas

tarefas. Entre as funções de gerenciamento de virtualização dos *hipervisores* estão armazenamento, virtualização de rede, virtualização de gerenciamento e a virtualização de serviços.

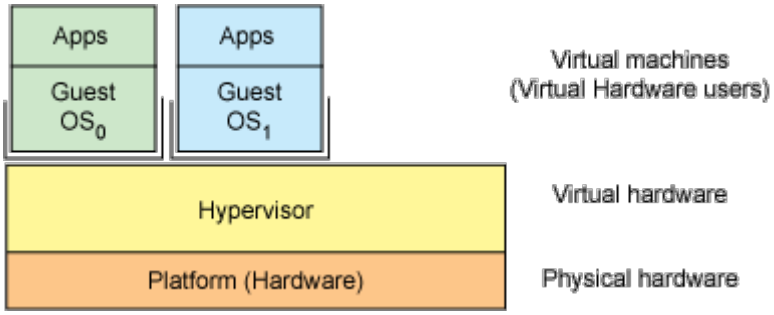

Figura 6 – Arquitetura simples do *hypervisor.* Fonte: Jones (2009).

Um *hypervisor* é um aplicativo que separa o *hardware* da máquina de cada convidado. Dessa forma, cada um vê uma máquina virtual em vez do *hardware* real. Em um nível superior, o *hypervisor* requer um pequeno número de itens para iniciar um sistema operacional convidado: uma imagem do sistema operacional para inicializar, uma configuração básica, como por exemplo, disco, dispositivo de rede, memória e processadores. O disco e o dispositivo de rede normalmente são mapeados para o disco físico da máquina e para o dispositivo de rede físico. Na Figura 7 ilustra-se esta relação.

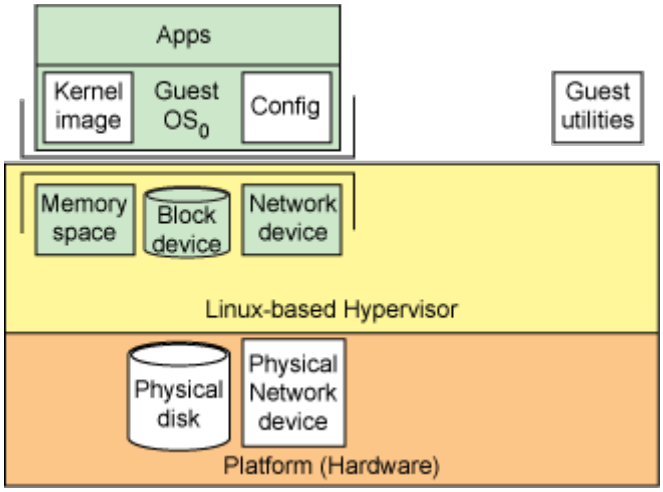

Figura 7 – Mapeamento de recursos no *hypervisor.* Fonte: Jones (2009).

O *hypervisor* mantém o isolamento de tarefas das máquinas virtuais, e é importante que todo mecanismo de controle, inclusive as interrupções sejam

controladas exclusivamente por ele. Desta forma se uma máquina virtual falhar é possível isolar o problema de forma que apenas ela será afetada, os outros sistemas continuarão seu funcionamento normalmente. Na Figura 8 é apresentada uma versão simplificada dos recursos de um *hypervisor* baseado em *linux.* É possível observar que todo mecanismo de entrada/saída, interrupções e mapeamento de páginas está dentro do *hypervisor,* que por sua vez faz a comunicação com o *hardware.*

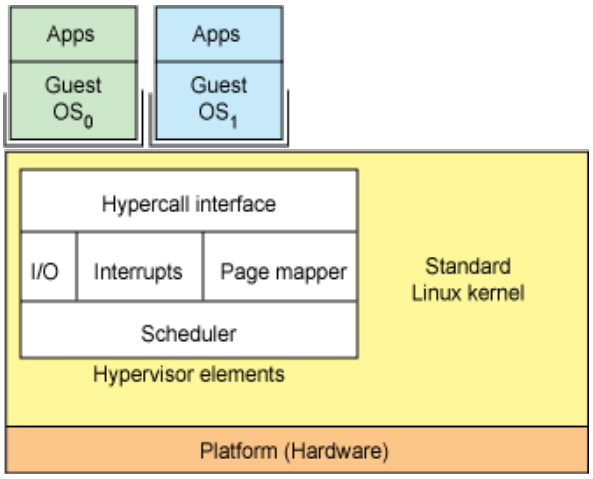

Figura 8 – Visualização de um *hypervisor.* Fonte: Jones (2009).

Um *hypervisor* deve ter pelo menos as seguintes propriedades (LAUREANO e MAZIERO, 2008):

· Equivalência: Provê um ambiente de execução quase idêntico ao da máquina real original. Dessa forma, todo programa executado em uma máquina virtual deve se comportar da mesma forma que o faria em uma máquina real;

· Controle de recursos: Deve possuir o controle completo dos recursos da máquina real, sendo que nenhum programa executado na máquina virtual deve possuir acesso a recursos que não tenham sido alocados a ele pelo *hipervisor*, que deve intermediar todos os acessos;

· Eficiência: As instruções do processador virtual devem ser executadas diretamente pelo processador da máquina real, sem intervenção do *hipervisor*. As instruções da máquina virtual que não puderem ser executadas pelo processador real devem ser interpretadas pelo *hipervisor* e traduzidas em ações equivalentes no processador real;

Cada máquina virtual é tratada de forma independente, cada uma com seus processadores virtuais, memória, discos e sistema operacional. Para o usuário final isto se torna transparente, ele nem tem e nem precisa ter conhecimento que a máquina que está utilizando é física ou virtual. Assim também é possível executar vários sistemas operacionais diferentes apenas em uma máquina física. A virtualização é, então, a principal tecnologia da computação em nuvem. Processadores, memória, disco e rede podem ser tratados como um conjunto de recursos e não mais como dispositivos separados e podem, desta forma, serem utilizados sob demanda (CHIEU *et al*., 2009).

Por conta desta flexibilidade e facilidade de alocação de recursos, a virtualização torna-se fundamental na computação em nuvem, provedores de serviços de nuvem utilizam virtualizadores para criar e gerenciar suas máquinas. Os recursos são apresentados ao gerenciador da nuvem e este gerencia e distribui os recursos de acordo com a demanda e oferta, porém no momento de criar as máquinas requisitadas ele utiliza um virtualizador que pode ser VmWare, Hiper-V, Xen, KVM ou outro.

As máquinas virtuais suportam configurações flexíveis e proporcionam controle da parcela do poder de processamento que podem consumir com base no tempo crítico da aplicação subjacente. O problema da inflexibilidade e dificuldade de remanejamento de recursos dentro de uma nuvem são resolvidos através do desenvolvimento de mecanismos de migração transparente de máquinas virtuais em todos os limites do serviço, com o objetivo de minimizar o custo do serviço (por exemplo, migrando para uma nuvem localizada em uma região onde o custo da energia é baixo) e os níveis de SLA são suficientes para a demanda. O gerenciador da nuvem monitora o uso de recursos e orquestra a migração dinâmica das máquinas virtuais com base no *feedback* de tempo real fornecido pelo sensor de serviço da nuvem (BUYYA et al. 2010).

A virtualização permite que servidores e dispositivos de armazenamento possam ser tratados em conjunto, de forma que possam ser alocados sob demanda (WEI; BLAKE, 2014). Desta forma, a virtualização é adaptada a

infraestrutura dinâmica que a nuvem exige, pois fornece vantagens no compartilhamento, gerenciamento e isolamento dos recursos (RIMAL et al.,2009).

Em Menken e Blokfijk (2010) são citados os principais benefícios obtidos com o uso da virtualização:

· Uso eficiente dos recursos: com a melhoria na tecnologia, os recursos de hardware não são totalmente utilizados permanecendo ociosos na maior parte do tempo. A virtualização torna a utilização dos recursos mais eficiente e reduzindo os custos operacionais e de gerenciamento;

· Redução dos custos com gerenciamento de recursos: O custo de manutenção e reposição para manter toda a infraestrutura física é muito alto, despende muitos recursos e gera muitos gastos para a empresa. Utilizando uma infraestrutura virtualizada, as empresas podem economizar e aproveitar melhor os recursos físicos disponíveis;

· Maior flexibilidade e portabilidade: o processo de ampliação das estações de trabalho e dos servidores é longo e custoso para as empresas, pois exige muito tempo na instalação e configuração das máquinas físicas, além da questão da ociosidade desses equipamentos. As máquinas virtuais podem ser facilmente instaladas e não despendem gastos adicionais com hardware, além de não requisitarem espaço físico extra;

· Isolamento: embora as máquinas virtuais possam compartilhar os recursos físicos de um único computador, elas ficam isoladas dos outros softwares da máquina física, incluindo outras máquinas virtuais, como se fossem máquinas físicas separadas;

· Eliminação de problemas com compatibilidade: Com a virtualização, vários sistemas operacionais podem ser instalados sobre uma camada de *software*, a qual faz uma interface com o *hardware* e elimina esses problemas de compatibilidade, sem que um sistema interfira nas configurações do outro.

2.6.1 Virtualização de *storage*

Outro problema que pode ser resolvido com a virtualização é o armazenamento. Com a virtualização de *storage* os discos possuem

gerenciamento centralizado e são distribuídos pela controladora que agrupa todo espaço disponível. Quando há uma solicitação de criação de disco, a mesma é feita diretamente à controladora que cria discos virtuais, desta forma um disco físico não precisa necessariamente ser dedicado apenas para um servidor.

Trabalhando desta forma, os discos são melhor aproveitados e é possível aumentar ou diminuir o espaço de um disco virtual conforme a necessidade.

Segundo Buyya (2011), a virtualização de *storage* consiste em abstrair armazenamento lógico a partir de um armazenamento físico. A partir do momento que o armazenamento de todos os dispositivos é centralizado em um centro de dados, torna-se possível a criação de discos virtuais. *Storages* são, geralmente organizados em *Storage Area Network (SAN* – rede de armazenamento*)* e conectados através de protocolos como o *Fibre Channel, iSCSI e NFS.*

2.6.2 Virtualização de redes

Buyya (2011) ainda cita que as redes virtuais permitem a criação de uma rede isolada para cada infraestrutura contratada. Uma *LAN* Virtual (*VLAN*) permite isolar o tráfego de uma rede específica dentro de uma rede física compartilhada com outros clientes, desta forma VMs podem ser agrupadas no mesmo domínio de *broadcast*. Na Figura 9 é mostrada esta configuração.

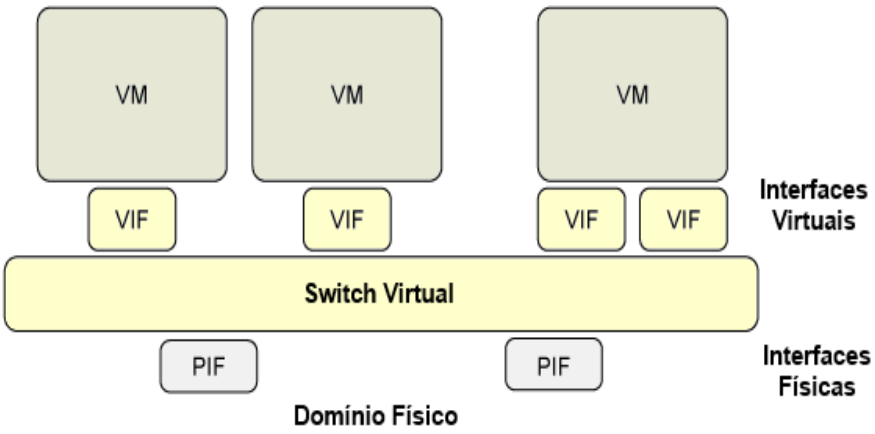

Figura 9 – Redes virtuais na Computação em Nuvem. (JONES, 2012)

Capítulo 3 Escalonamento de Recursos

Nos serviços de computação em nuvem há os que possuem os recursos e disponibilizam estes recursos para uso, há também os que utilizam os recursos (consumidores). Os dois tipos envolvidos na nuvem podem estar geograficamente distribuídos e em zonas de tempos diferentes.

Para facilitar o gerenciamento e distribuição o ideal é que sejam gerenciados de forma centralizada. Desta forma o gestor da nuvem pode escalonar recursos de acordo com a necessidade do consumidor e disponibilidade dos fornecedores, inclusive utilizando a localização, em alguns casos, para determinar qual fornecedor atenderá a demanda do consumidor. Também como uso de *CPU*, memória, *throughput* e disco que sejam suficientes para atender a demanda cumprindo o SLA contratado e com QoS.

O consumidor solicita recursos de acordo com sua necessidade, e pode negociar valor para processamento baseado na demanda, no valor absoluto, na prioridade, ou no orçamento previsto (BUYYA et al.,2002).

Desta forma o provedor pode cobrar de acordo com a quantidade de recursos solicitados de acordo com o tempo de utilização como também há a possibilidade de temporariamente aumentar ou diminuir recursos e cobrar proporcionalmente.

Vieira (2016) relata duas formas de cobrança mais utilizados atualmente: o*n-demand e reserved*. O modelo de cobrança *reserved* exige o pagamento de uma taxa inicial para reservar a instância da máquina virtual. Após reservada, o usuário obtém um desconto no momento em que utilizá-la. Esta estratégia pode ser vantajosa para o usuário apenas se ele tem a expectativa de utilizar a máquina por um longo tempo. Já para a modelo *on-demand*, o usuário só paga pelo que utilizar, no momento que utilizar. Quando a demanda é para curto prazo, este modelo se torna mais viável.

Um modelo de negócio bastante conhecido e utilizado é o *pay-per-use*. Nele, os clientes pagam pelos recursos que utilizam. Porém, neste modelo é possível que o cliente pague por recursos que não estão sendo utilizados. Como ele alocou determinada instância de máquina virtual, enquanto esta

instância estiver ativa ele paga pelo recurso, mesmo que não esteja utilizando (BUYYA, 2011).

Ao longo dos últimos anos, as tecnologias da computação em grade progrediram para um paradigma orientado para o serviço que permite uma nova forma de provisionamento de serviços com base em modelos de computação conforme o uso. Os usuários consomem esses serviços com base nos requisitos de QoS. Em tais redes de *pay-per-use*, o custo de execução do fluxo de trabalho deve ser considerado durante o agendamento baseado em restrições de QoS (BUYYA 2011).

Uma forma de solucionar este problema foi proposto por Ibrahim, He e Jin (2011), um modelo de negócio chamado *pay-as-you-consume*, que cobra dos clientes aquilo que eles realmente utilizaram excluindo interferências de outras máquinas virtuais. A ideia principal é um mecanismo de aprendizagem baseado no modelo de previsão de custo relativo.

Os provedores de infraestrutura como serviço oferecem períodos de cobrança com diferentes granularidades, subprocessos ou tarefas. Essas tarefas podem então ser distribuídas, por exemplo, na *Amazon* EC2 por hora, enquanto a Microsoft *Azure* a tarifa é mais flexível e pode ser cobrada por minuto (Rodriguez, Buyya 2017).

3.1 Uso e Gerenciamento de Recursos na Computação em Nuvem

Além de todas questões técnicas envolvidas na escolha e configuração da nuvem adequada, há a questão do gerenciamento de recursos.

Por conta da facilidade de contratação, é possível que a companhia perca o controle se o que foi contratado é realmente necessário. Por causa disto houve uma explosão de contratações de serviços na nuvem e na maioria das vezes superdimensionado o que fez a conta começar a ficar muito alta o que era para ser um modelo mais econômico começou a perder o sentido. Neste caso há um desperdício de recursos no provedor de serviços e a companhia acaba pagando por algo que não necessita (Vieira 2016).

Outro fator importante é o que será levado para nuvem pública e o que ficará na nuvem privada. De acordo com Armbrust et al. (2010) é certo hoje que o modelo híbrido será o mais utilizado, a dificuldade muitas vezes é definir critérios para estas definições. O gerenciamento desta nuvem também pode ser um grande desafio, envolve, além dos contratos de nuvem pública, a gestão e configuração da nuvem privada.

3.2 Escalonamento de Recursos na computação em nuvem

Considerando a dificuldade em especificar a quantidade de recursos necessários para determinada execução e também em medir a utilização de recursos para que o valor cobrado seja justo, faz-se necessário a utilização de ferramentas capazes de auxiliar o administrador a gerir sua nuvem, seja ela pública, privada ou híbrida. Umas dessas ferramentas são os escalonadores de recursos que tem o propósito de auxiliar na escolha da configuração ideal para determinada necessidade.

Pode-se definir o escalonamento como o mapeamento de recursos computacionais para a execução de atividades independentes, determinando a utilização de recursos para satisfazer a necessidade de determinada tarefa de acordo com orçamento disponível. Tanembaum (2008) explica que o escalonador é responsável por decidir qual tarefa executará primeiro, caso haja várias tarefas prontas para executar. Decide também qual é o processador mais adequado para executar a tarefa selecionada e decidir qual é o intervalo de tempo que cada tarefa executará em cada processador. A meta é que o escalonador conseguirá dividir as tarefas e os recursos disponíveis de forma a concluir as tarefas no menor tempo possível.

Baseado em técnicas que consideram métricas de uso, seu propósito é garantir configurações que atenderão a necessidade sem desperdiçar recursos, garantindo SLA e Qos das requisições. Os escalonadores selecionam máquinas virtuais com diferentes desempenhos e preços. Quanto mais informações possam ser fornecidas para configurar o escalonador melhor será definido o tipo de máquina a ser utilizada. Se o objetivo for concluir a tarefa no

menor tempo possível, serão escolhidas as máquinas mais rápidas. Porém, se não for possível pagar por estes recursos, o escalonador pode optar por máquinas mais lentas (e mais baratas) para se adequar ao orçamento disponível.

Segundo Prajapati e*t al* (2013), o escalonamento é um cenário de equilíbrio em que processos ou tarefas são agendados de acordo com os requisitos fornecidos. Na computação em nuvem, os algoritmos de escalonamento de VM são usados para agendar os pedidos de VM para máquinas físicas do *data center* conforme requisitado e preenchido com os recursos solicitados (ou seja, memória *RAM*, disco, processador e largura de banda). Há muitos fornecedores de nuvem no mercado que têm centros de dados com capacidades diferentes e variadas configurações de máquinas físicas disponíveis.

As nuvens infraestrutura como serviço oferecem um acesso fácil, flexível e escalável com o uso de algoritmos de escalonamento, uma heurística que decide o tipo e infraestrutura para a implantação de fluxos de trabalho científicos de grande escala, número de máquinas virtuais a utilizar e quando contratar e quando liberá-las. Eles permitem que os sistemas de gerenciamento de fluxo de trabalho acessem uma infraestrutura compartilhada e decidam qual o melhor local para executar determinado fluxo (Rodriguez, Buyya 2017).

Ainda conforme Rodriguez e Buyya (2017), quando comparada a outros sistemas de distribuição, por exemplo, grade, a computação em nuvem oferece mais controle sobre tipo e quantidade de recursos. Com a flexibilidade e abundância de recursos, surge a necessidade de estratégias que trabalhem em conjunto com algoritmos de escalonamento.

Na Figura 10 Rodriguez e Buyya (2017) propoem uma arquitetura básica para escalonamento de fluxos de trabalhos na computação em nuvem.

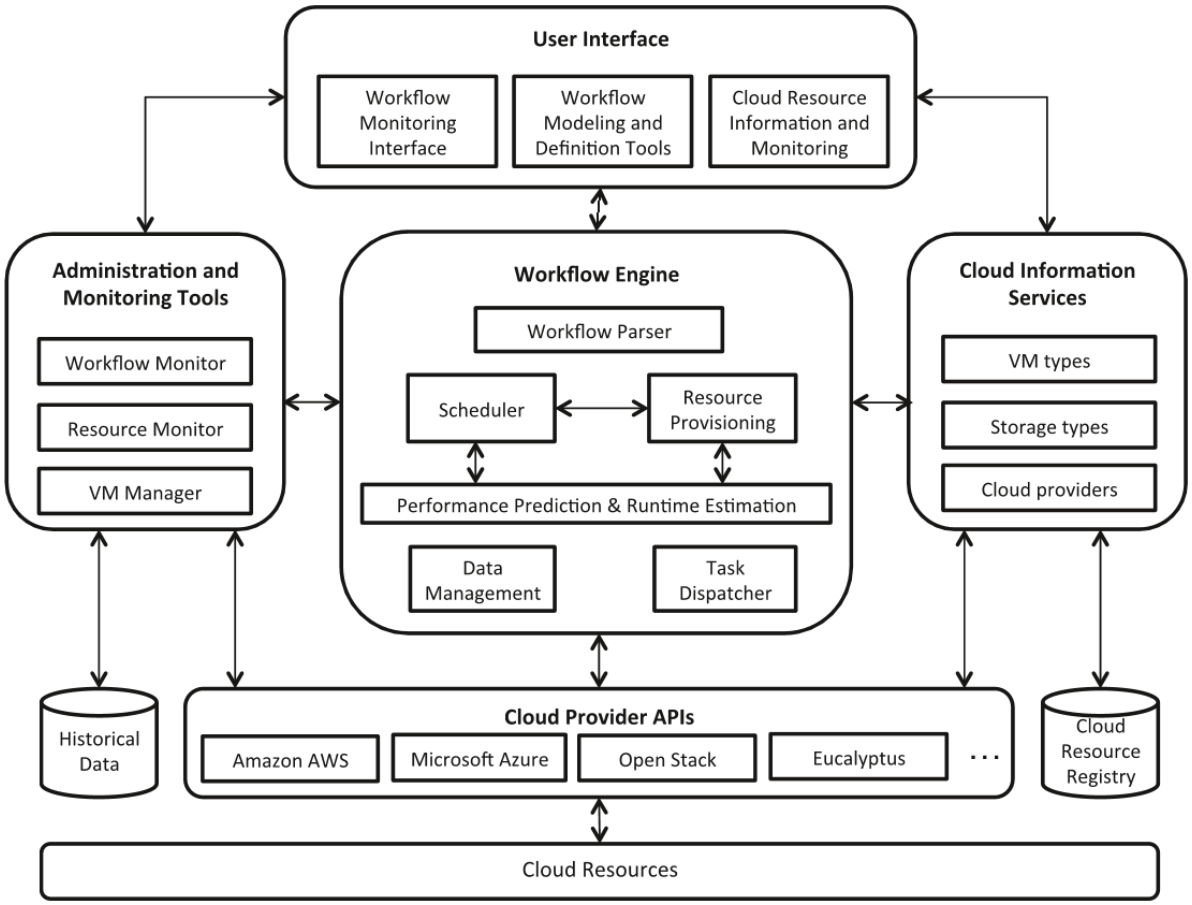

Figura 10: Arquitetura de referência para gerenciamento de fluxos de trabalho (Rodriguez e Buyya 2017)

U*ser interface:* Fornece ambiente para os usuários criarem, editarem, submeterem e monitorarem suas aplicações.

Workflow engine: É o centro do processo e é responsável por gerenciar a execução atual e fazer todo controle das cargas de trabalho com os recursos disponíveis. É responsável por selecionar e provisionar os recursos de nuvem, escalonando de acordo com os recursos disponíveis, cumprindo as políticas baseadas em QoS.

Administration and monitoring tools: As ferramentas de administração e monitoramento habilitam o dinâmico e contínuo monitoramento dos fluxos de trabalho, bem como também o gerenciamento da *performance* do sistema.

Cloud information services: Este componente provê para o *workflow engine*, informações a respeito de outros provedores de nuvem, suas

características, recursos, preços, localização e outras informações pertinentes que auxiliem na escolha da infraestrutura correta para alocar uma tarefa.

Cloud provider APIs: Esta API fornece a integração entre as aplicações e os serviços de nuvem. Através dela pode-se alocar uma máquina virtual ou desalocar um recurso anteriormente alocado.

Casavant e Kuhl (1988) criaram uma classificação para os diferentes tipos de escalonamento, estes podem ser:

Local: atribuição de tarefas por *slots* de tempo.

Global: Executam as tarefas em um conjunto de processadores.

Estático: As informações sobre as tarefas seus relacionamentos com outras tarefas são totalmente conhecidas antes da execução do programa (EL-REWINI; ABD-EL-BARR, 2005).

Dinâmico: Escalonamento não-determinístico, algumas informações podem não ser conhecidas antes da sua execução (EL-REWINI; ABD-EL-BARR, 2005).

Ótimo: Quando é possível determinar todas as informações sobre a tarefa.

Subótimo: Quando não é possível determinar as informações das tarefas, mas a solução é aceitável.

Aproximado: Quando uma boa solução é encontrada, neste caso dispensa-se a ótima para determinado problema.

Heurística: Suposições realistas a respeito das características dos processos e de carga do sistema.

Não distribuído: a responsabilidade do escalonamento de tarefas reside em um único processador.

Distribuído: o trabalho envolvido na tomada de decisões deve ser distribuído entre os processadores.

Não cooperativo: processadores tomam decisões individualmente, sem levar em conta o efeito que pode ter em todo sistema.

Cooperativo: Ao contrário do anterior, neste caso, todos componentes cooperam entre si.

Adaptativo: Mudam dinamicamente de acordo com o comportamento anterior para se ajustar as novas condições do sistema.

Não adaptativo: Não existe alteração, não leva em conta a execução e a mudança dinâmica de parâmetros.

Balanceamento de carga: Busca o equilíbrio entre os processadores do sistema.

Bidding: Cooperação entre os nós que executarão as tarefas, troca de informações para uma melhor execução das tarefas.

Probabilístico: Processos são escolhidos aleatoriamente, e entre esses processos, os que apresentam os melhores resultados são escolhidos.

Na Figura 11 são ilustradas estas classificações.

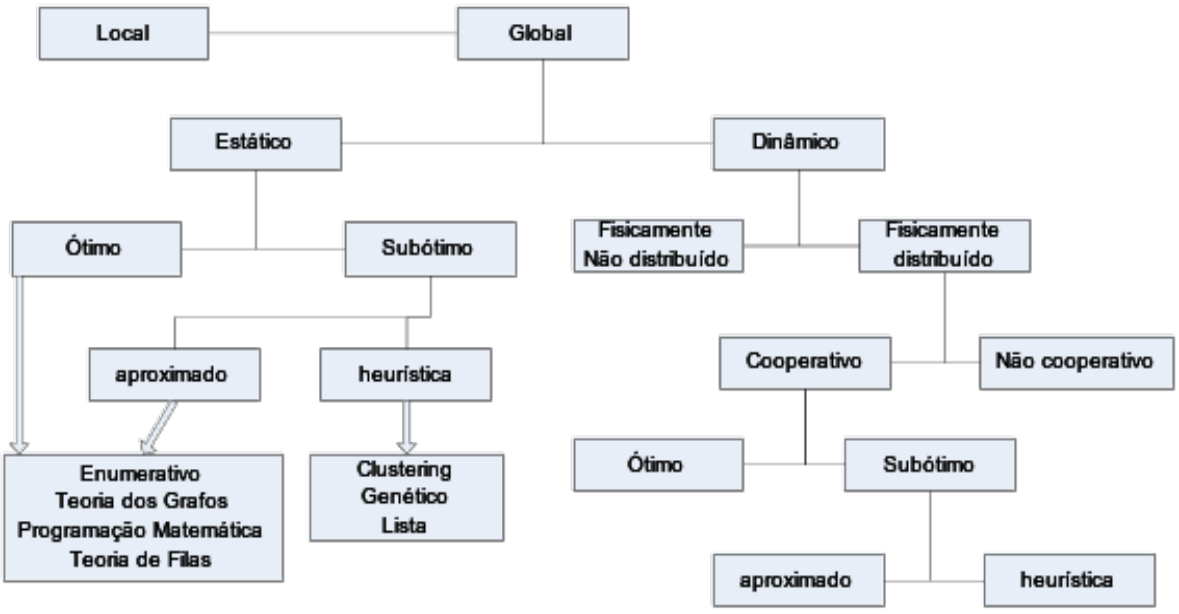

Figura 11 – Classificação do escalonamento - CASAVANT (1988)

3.3 Algoritmos

Nesta seção são apresentados algoritmos utilizados no escalonamento de recursos na computação em nuvem. Também uma breve explicação sobre cada um deles relacionando com trabalhos existentes. Particularmente, o algoritmo *particle swarm* será aprofundado e com mais citações de trabalhos pois é utilizado como base para este trabalho.

3.3.1 Min-Min e Max Min

Os algoritmos Min-Min e Max-Min, (Maheswaran, 1999) utilizam heurística que atribui a prioridade de escalonamento de acordo com o tempo de execução estimado. Todas atividades são listadas e mapeadas, estas são associadas ao tempo de execução observados ou previstos em cada um dos recursos disponíveis. No Min-Min, após a lista pronta, a atividade que tiver o menor tempo de execução é selecionada e mapeada para o recurso no qual o menor tempo foi obtido. Após isso são retiradas do mapeamento e o processo é novamente realizado até que todas sejam escalonadas. No algoritmo Max-Min, a prioridade é para atividades que têm o maior tempo na lista de tempos de execução. O restante do processo é feito como no Min-Min. Mapear uma atividade com maior tempo de execução para o melhor recurso primeiro permite que esta atividade seja executada concorrentemente com as demais atividades, que têm tempo de execução mais curtos. No trabalho de Khan (2014), foi analisado o algoritmo Min-Min para realizar o balanceamento de carga em computação em nuvem.

Na Figura 12 é apresentado fluxo básico de ações do algoritmo, começando as etapas primeiro com a estratégia Min-Min.

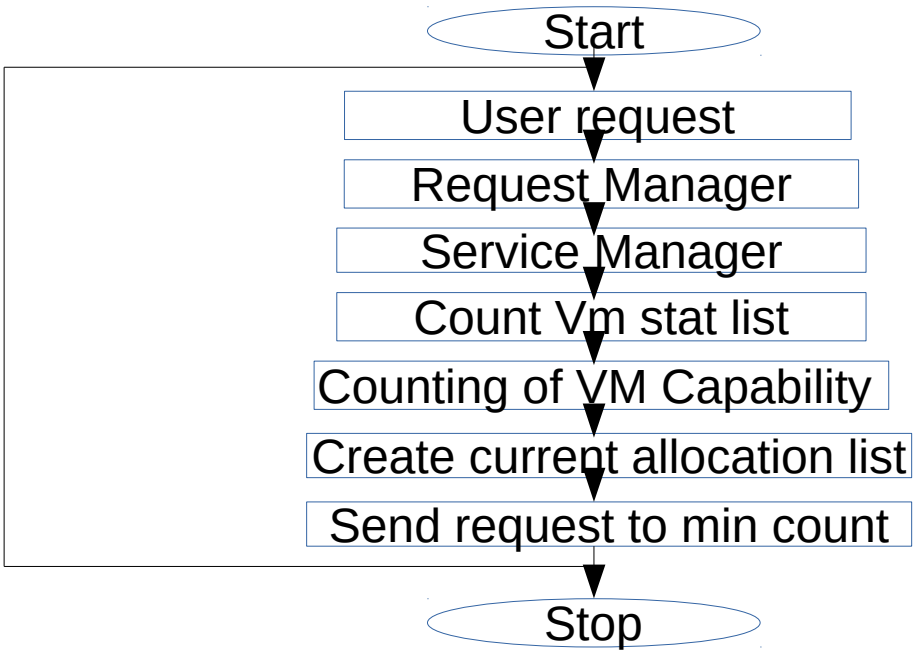

Figura 12 – Fluxo Min Min (KHAN, 2014)

O algoritmo identifica pela primeira vez a VM com tempo de execução menor e o recurso disponível. Assim, a VM com tempo de execução menor será agendada primeiro. Depois disso, considera o tempo mínimo de conclusão, uma vez que alguns recursos estão agendados com alguma outra VM.

Na segunda rodada escolhe os recursos com carga maior e reatribui os recursos com carga menor. O tempo de conclusão dessa tarefa é calculado para todos os recursos na programação atual. Então o tempo máximo de conclusão dessa VM é comparado com o *makespan(*tempo total de execução*)* produzido pelo Min-Min. Se for menor do que *makespan*, a tarefa é reprogramada para o novo recurso, e o tempo de preparação de ambos os recursos é atualizado, caso contrário, o próximo com tempo máximo de conclusão dessa tarefa é selecionado e as etapas são repetidas novamente.

O processo para se todos os recursos de todas as VM atribuídas neles foram considerados para reagendamento. Assim, os recursos possíveis são reprogramados nos recursos inativos ou com mínimo de carga.

O algoritmo proposto foi implementado usando simulador de nuvem "*CloudSim*", "*GridSim*" e "*JavaSim*". A estrutura é mostrada na Figura 13.

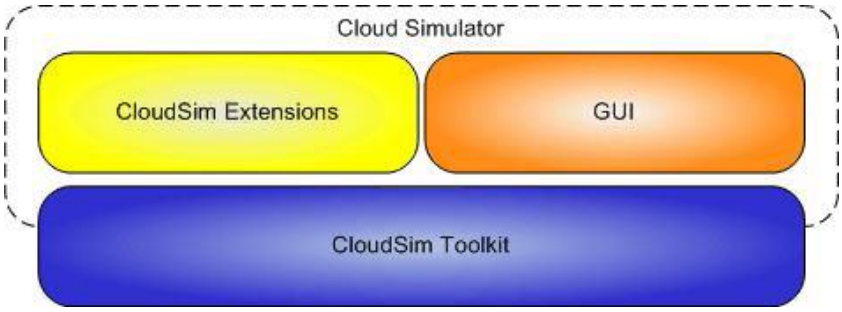

Figura 13 - Analisador de nuvem *kit* ferramentas do Cloud-Sim Khan (2014)

3.3.2 Algoritmo Genético

Em Yu e Buyya (2006), o objetivo é otimizar custo e tempo de execução. Não foi considerado apenas tempo de execução mas também custo monetário, confiabilidade e segurança. O agendamento do fluxo de trabalho centra-se no mapeamento e gerenciamento da execução de tarefas interdependentes em

diversos serviços. Em geral, o problema das tarefas de mapeamento em serviços distribuídos pertence a uma classe de problemas “de soluções difíceis” (YU e BUYYA, 2006). Para tais problemas, nenhum algoritmo conhecido é capaz de gerar a solução ideal dentro do tempo polinomial. Embora o problema do agendamento do fluxo de trabalho possa ser resolvido usando uma busca exaustiva, a complexidade dos métodos para solucioná-lo é muito grande.

Para fornecer técnicas de pesquisa robustas que permitam aliar solução de alta qualidade com um grande espaço de busca em tempo polinomial, aplicando o princípio da evolução foi utilizada a base de um algoritmo genético, que tem a capacidade de combinar estas melhores soluções.

Nos algoritmos genéticos, qualquer solução no espaço de busca do problema é representada por um indivíduo. Um algoritmo genético mantém uma população de indivíduos que evolui ao longo das gerações. O valor da aptidão indica quão bom o indivíduo é comparado com outros na população. Um algoritmo genético típico consiste nas seguintes etapas: (1) criar uma população inicial constituída por soluções geradas aleatoriamente. (2) Gerar nova população através da aplicação de operadores genéticos, por exemplo, seleção, crossover e mutação, um após o outro. (3) avaliar o valor da aptidão de cada indivíduo na população. (4) repetir os passos 2 e 3 até o algoritmo convergir para a melhor solução.

Para o problema de agendamento do fluxo de trabalho, foi necessário atender as seguintes condições: (1) A tarefa só pode ser iniciada após a conclusão de todos os antecessores. (2) Cada tarefa aparece apenas uma vez no cronograma. (3) Cada tarefa deve ser alocada para um espaço de tempo disponível de um serviço capaz de executar a tarefa. Cada indivíduo na população representa uma solução viável para o problema e consiste em um vetor de tarefa de atribuições. Cada atribuição de tarefa inclui quatro elementos: ID de tarefa, ID de serviço, hora início e hora final. ID de tarefa e ID de serviço identificam a qual serviço cada tarefa é atribuída, hora início e hora final indicam o intervalo de tempo alocado para a execução da tarefa. No entanto, o envolvimento de prazos durante a operação genética pode levar a uma situação muito complicada, porque qualquer mudança feita em uma tarefa

pode exigir o ajuste dos valores de hora início e hora final de suas tarefas sucessivas. Neste caso os prazos foram ignorados. As cadeias de operação codificam apenas a alocação de serviço para cada tarefa e a ordem da alocação de tarefas em cada serviço. Após o cruzamento e mutações, um método de atribuição de intervalo de tempo é aplicado para transferência de uma cadeia de operação para uma programação viável.

Uma função de *fitness* é usada para medir a qualidade dos indivíduos na população de acordo com o dado objetivo de otimização. Como o objetivo do agendamento é minimizar o desempenho com base em dois fatores, o tempo e custo monetário, a função de *fitness* separa a avaliação em duas partes: custo e tempo. As funções usam duas variáveis binárias, α e β. Se os usuários especificarem uma restrição de orçamento, então α = 1 e β = 0. Se os usuários especificarem um prazo, então α = 0 e β = 1.

Em algoritmos genéticos também é utilizada a técnica de mutação, que consiste em filhos adquirir qualidades que não estão presentes em nenhum dos pais. Desta forma é possível gerar um novo e melhorado material genético. Neste algoritmo foram utilizadas duas técnicas de mutação: *swapping mutation e replacing mutation*. *Swapping mutation* visa alterar a ordem de execução das tarefas em um indivíduo que disputa pelo mesmo tempo. É implementado da seguinte forma: (1) Um serviço no indivíduo é selecionado aleatoriamente. (2) As posições de duas tarefas independentes são selecionadas aleatoriamente no serviço e são trocadas. R*eplacing mutation* visa reatribuir um serviço alternativo a uma tarefa em um indivíduo. É implementado desta forma: (1) Uma tarefa é selecionada aleatoriamente no indivíduo. (2) Um serviço alternativo que é capaz de executar a tarefa é selecionado aleatoriamente para substituir a alocação da tarefa atual.

3.3.3 Round Robin

Schmitt (2016) apresenta o algoritmo Round-Robin que executa um processo em uma determinada fração de tempo, após esse tempo acabar ele parte para o próximo processo na fila, este processo se repete até que não haja mais nada na fila.

É o tipo de escalonamento preemptivo mais simples e consiste em repartir uniformemente o tempo da CPU entre todos os processos que estejam prontos para execução. Os processos são colocados em uma fila circular, com objetivo de alocar uma fatia igual de tempo da CPU dentro de um tempo determinado. Caso um processo não termine dentro de sua fatia de tempo, ele é colocado no fim da fila e uma nova fatia de tempo é alocada para o processo no começo da fila.

Configurado de uma forma muito simples, mas pode trazer problemas se os tempos de execução são muito diferentes entre si. Quando existirem muitas tarefas ativas e de longa duração no sistema, as tarefas curtas terão o seu tempo de resposta degradado, pois as tarefas longas reciclarão continuamente na fila circular, compartilhando de maneira equitativa a CPU com tarefas curtas.

É definida uma pequena unidade de tempo. Todos os processos são armazenados na fila que é circular. O escalonador da CPU percorre a fila, alocando a CPU para cada processo durante o período de tempo definido. Mais precisamente, o escalonador retira o primeiro processo da fila e inicia sua execução. Se o processo não termina durante o tempo definido ele é então retirado da execução e inserido no fim da fila. Se o processo termina antes do tempo definido, a CPU é liberada para receber novos processos.

No trabalho de Schmitt (2016), o algoritmo de *round robin* foi melhorado, é mais ideal e aloca os trabalhos para os recursos mais adequados, baseados nas informações da VM, como sua capacidade de processamento, carga nas máquinas virtuais e duração das tarefas com a sua prioridade. A programação estática desse algoritmo usa a capacidade de processamento das VMs, o número de entradas das tarefas e o comprimento de cada tarefa para decidir a alocação apropriada.

O agendamento dinâmico (em tempo de execução) desse algoritmo, usa a carga em cada uma das VMs juntamente com as informações mencionadas anteriormente para decidir a alocação da tarefa para a VM apropriada. Existe uma probabilidade em tempo de execução que, em alguns casos, a tarefa pode levar uma execução mais longa do tempo do cálculo inicial devido à execução

de vários números de ciclos (como um *loop*) nas mesmas instruções baseadas nos complicados dados de tempo de execução.

Em tais situações, o balanceador de carga resgata o planejamento, controla e reorganiza os trabalhos de acordo com o *slot* disponível nas outras VMs não utilizadas/ subutilizadas, movendo um trabalho de espera das VM altamente carregadas. O balanceador identifica as VMs não utilizadas/subutilizadas através do recurso sempre que uma tarefa for concluída em qualquer uma das VMs. Se não houver VM não utilizada, então o balanceador de carga não assumirá qualquer migração de tarefas entre as VMs. Se encontrar qualquer VM não utilizada/subutilizada, então ele migrará a tarefa da VM sobrecarregada à VM não utilizada/subutilizada.

O balanceador nunca examina a carga do recurso (VM) de forma independente, a qualquer momento para contornar a sobrecarga nas VMs. Isso ajudará reduzir o número de migrações de tarefas entre as Vms e o número de execuções da sonda de recurso nas Vms.

A implementação consiste em 5 passos:

Agendador estático: Tem a função de encontrar o máximo de VMs adequadas e atribui as tarefas às VMs com base nos algoritmos (simples *round robin*, *round robin* ponderado e *round robin* melhorado) aplicado no planejador.

Agendador dinâmico: Tem a função de colocar a execução de trabalhos de chegada para as máquinas virtuais mais adequadas com menor uso naquela hora da chegada específica do trabalho.

Balanceador de carga: Decide sobre a migração da tarefa de uma máquina virtual altamente carregada para uma máquina virtual ociosa ou VM carregada em tempo de execução sempre que encontrar uma máquina virtual ociosa ou menos carregada utilizando as informações do monitor de recursos.

Agendador independente de tarefas: Identifica se tem tarefas filhas se tiver, coloca em uma fila de dependência. Se não tem tarefas filhas, ou se já executou todas tarefas filhas, é utilizado o agendador dinâmico/estático.

Monitor de recursos: Se comunica com todas as fontes de recursos das VM e coleta as capacidades de cada uma, carga atual e número de trabalhos em filas de execução / espera em cada VM para decidir quais

máquinas virtuais apropriadas para os trabalhos. A tarefa do estimador de necessidades identifica o comprimento das tarefas para ser executado e transfere os resultados estimados para a carga do balanceador para as escolhas operacionais. A Figura 14 apresenta a arquitetura do algoritmo. Na Figura 15 é apresentado o fluxograma básico do agendador de tarefas.

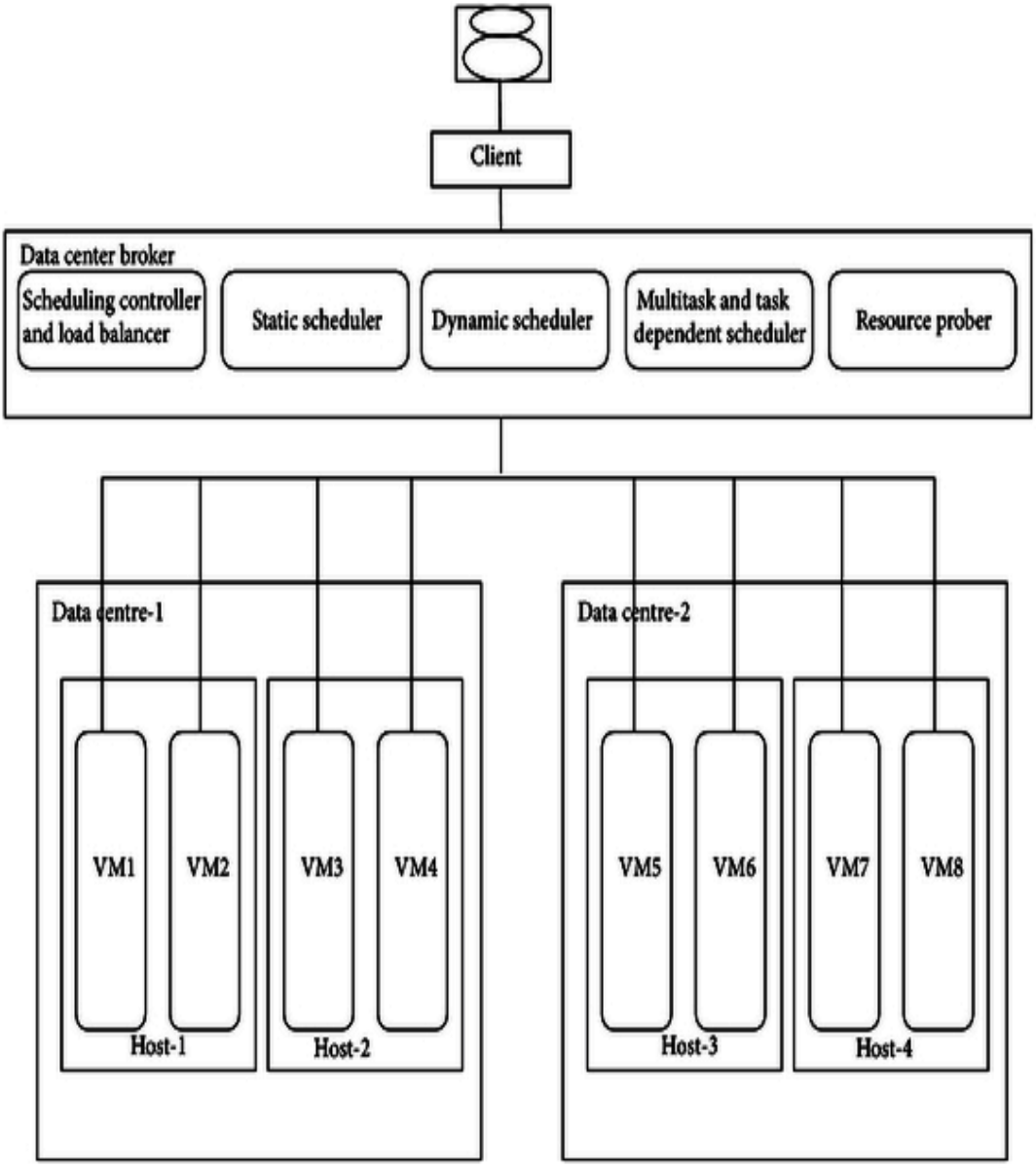

Figura 14 – Arquitetura do algoritmo (DEVI 2016)

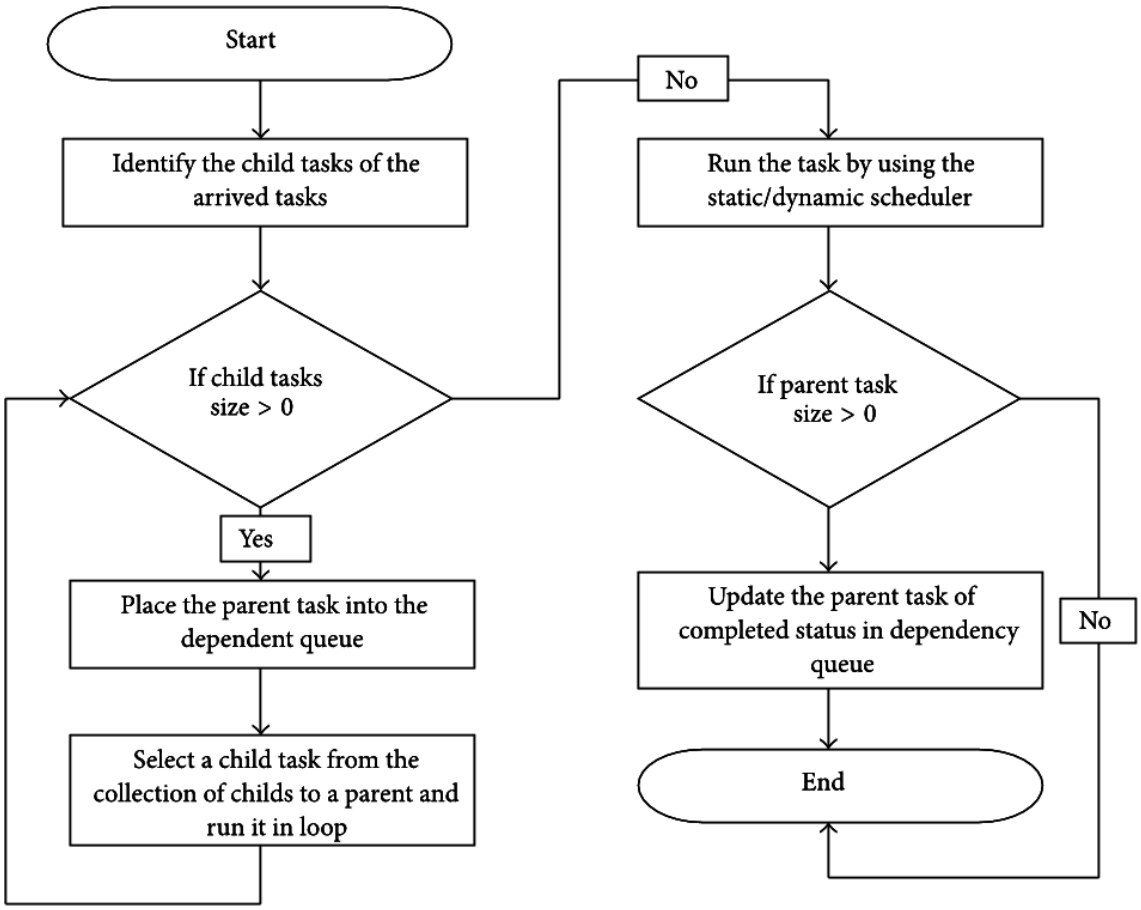

Figura 15 – Agendador independente de tarefas (DEVI 2016)

3.3.4 *Particle Swarm*

Nesta seção será apresentado o algoritmo *Particle Swarm*, que é a base deste trabalho (YUAN, 2014).

Também são apresentados dois trabalhos que utilizaram o algoritmo para resolver o problema de escalonamento de recursos em computação em nuvem. Há uma breve explicação e são apresentados seus resultados.

O algoritmo *particle swarm* é um algoritmo inspirado em padrões da natureza, inspirado pelo comportamento social e cooperativo exibido por várias espécies por forma a realizar as suas necessidades no espaço de pesquisa.

O algoritmo busca encontrar a melhor solução no campo de busca, para isso cria um grupo de partículas que se movimentam no espaço no decorrer do

tempo, guia-se pela melhor localização local de cada partícula e pela melhor localização global do grupo de partículas.

A fórmula do algoritmo é demonstrada pela Equação 1:

$$^{V}k+1 = {}^{w}{}^{v}k + {}^{c}1{}^{r}1(Pbest_k - {}^{x}k) + {}^{c}2{}^{r}2(gbest - {}^{x}k)$$

$$^{x}k + 1 = {}^{x}k + {}^{v}k$$

eq. 1 Representação do algoritmo *particle swarm*

As variáveis $^{V}k+1$ são, velocidade e posição da partícula.

- W é o coeficiente de inércia.
- $Pbest_k$ é a melhor posição conhecida da partícula k
- $gbest$ melhor posição conhecida dentre todas as partículas
- $^{c}1$ e $^{c}2$ constantes de aceleração referentes ao melhor individual e global, respectivamente.
- $^{r}1$ e $^{r}2$ números aleatórios extraídos do intervalo [0,1]

As etapas do algoritmo são:

1 Iniciar uma população X de indivíduos de dimensão Y. Essa inicialização deve ser um vetor bidimensional, cada linha representa uma partícula e com número de colunas de acordo com os nós do problema a ser resolvido.

2 Determinar os valores das constantes de velocidade e pesos do valor global e valor local.

3 Calcular próxima posição.

4 Verificar se o critério de parada foi atingido, e esse critério pode ser um valor pré-determinado ou número de iterações. Se sim, finalizar o algoritmo.

5 Sortear os números aleatórios para $^{r}1$ e $^{r}2$.

6 Determinar a melhor posição global e individual.

7 Atualizar a velocidade das partículas.

8 Atualizar as posições das partículas.

9 Voltar para o item 3.

Em Yuan (2014) foi realizada uma otimização do algoritmo *particle swarm* para resolver problema de otimização de recursos para máquinas virtuais em computação em nuvem.

O algoritmo consiste em $T = \{t_1, t_2, ..., t_n\}$ representando as tarefas que esperam para ser agendada em uma unidade de tempo, n é o número de tarefas. $N = \{n_1, n_2, ..., n_m\}$ representa o conjunto de nós do sistema da nuvem, assumindo que o sistema da nuvem tem m nós.

Para o sistema de computação em nuvem, n(i) representa os recursos computacionais; Para o sistema de armazenamento em nuvem, n(i) representa os dados em n(i). $V = [v_1, v_2, ..., v_n]$ representa o agendamento de tarefas. Para o sistema de armazenamento em nuvem, v(i) representa a i-ésima tarefa de dados e é fornecida por nós de recursos que representam o valor v, e o comprimento do vetor é a quantidade total de agendamento de tarefas por unidade de tempo. Por exemplo, um vetor de agendamento de tarefas [5, 1, 3, 2, 1, 6], o comprimento deste vetor é 6 e representa necessidade do número de tarefas por unidade de tempo. O valor que, com base na posição de nº 1 é 5, representam os dados de a tarefa 1 é proporcionada pelo nó do sistema 5. Assim, os dados da tarefa 2 e 5 são fornecidos pelo nó 1; os dados da tarefa 3 são fornecidos pelo nó 3; os dados de A tarefa 4 é fornecida pelo nó 2; os dados da tarefa 6 são fornecidos pelo nó 6. Para o sistema de computação em nuvem, é em nome de uma tarefa colocada em um nó.

No modelo matemático adotado para o *Particle Swarm* (enxame de partículas) é gerado aleatoriamente um número de partículas em um determinado espaço de busca. Dentro deste espaço, cada partícula corresponde a uma possível solução, representada por sua posição no espaço de busca para um dado problema.

As partículas têm associado um valor de velocidade e também realizam um deslocamento sob a ação de três vetores que se somam. Essas influências são a inércia, memória e comparação. O primeiro vetor impele a partícula em uma direção idêntica à que ela vinha seguindo. A memória atrai a partícula na direção da melhor posição até o momento, ocupado pela partícula dentro da

sua vida. O último vetor atrai a partícula na direção do melhor ponto do espaço até o momento, descoberto pelo “enxame”.

Na implementação do algoritmo, o primeiro passo é gerar as N partículas que formarão o *swarm* ou o “enxame” com suas respectivas posições. Pode-se também neste momento, arbitrar velocidades iniciais para cada partícula. O algoritmo manter-se-á ativo, atualizando os vetores de velocidade e posição ciclicamente até que seja atingido qualquer critério de parada (número máximo de iterações atingido ou partícula com aptidão desejada).

A função *fitness* assume que um sistema de nuvem é uma matriz m x m, representa a largura de banda entre dois nós, e assume que é provável que haja mais de uma caminho, o cálculo da função fitness é definido por:

$$F(_{i,\,j}) = \max(\sum_i \frac{b_{ij}}{t_i})$$

N qual i representa o número da tarefa, j representa o número de nó de recurso, b_{ij} é largura de banda entre o nó i e nó j. A função faz avaliação se os resultados são bons ou maus para determinada partícula.

Para garantir a diversidade do esquema de agendamento, o cálculo das condições seguintes da iteração é:

(a) Vetor de posição das partículas iniciais:

O grupo é definido dentro de um determinado intervalo; na iteração, se o processo exceder o alcance, surge a necessidade de aplicar as regras e reavaliar.

(b) O vetor de velocidade da partícula inicial é limitado dentro de um certo intervalo, no processo de iteração se excede esse alcance, a necessidade de aplicar novamente as regras.

A otimização proposta é mostrada pela melhoria na função de cálculo da velocidade e posição, que ficou é mostrada na Equação 2:

$$V_i + 1 = \omega \times V_i + c\,1 \times D\,1 + c\,2 \times D\,2$$

$$X_i + 1 = X_i + V_i + 1$$

Eq. 2 – Melhoria do algoritmo *particle swarm*

Onde r1 e r2 são números randômicos entre 0 e 1:

$D\,1 = r\,(X_p - X_i)$, $D\,\min \leq D\,1 \leq D\,\max$

$D\,2 = r\,(X_g - X_i)$, $D\,\min \leq D\,2 \leq D\,\max$; c 1 e c 2 são fator de aprendizagem, no qual c1 representa a habilidade cognitiva da partícula de ir para a melhor posição e direção. Já c2 representa a habilidade do grupo em ir para a melhor posição e direção. O item ω representa o coeficiente do peso que a partícula mantem o movimento de inércia, reflete a influência da última iteração na velocidade atual, Equação 3.

$$\omega = (D\,\max - D\,\min) \times i \div L$$

$$D\,\max + D\,\min$$

Eq. 3 Coeficiente de peso da partícula

Onde i é o número atual de iterações, L é o total do número de iterações, se ω é o maior valor, o algoritmo realiza uma busca global da capacidade; se ω for o menor valor, o enxame de partículas o algoritmo tem uma tendência para a busca local.

Quando a diferença entre a posição atual do indivíduo e a posição ideal são grandes, especialmente após a liberação da partícula, o algoritmo então limita a velocidade da partícula. Se a velocidade da partícula é excessiva pode fazer com que a partícula caminhe para o global muito rápido. O limite de velocidade também pode evitar tal situação, e aumentar a taxa de convergência, também reter a liberação de recursos de pesquisa de partículas para que ele possa convergir para o caminho ideal global ao ser lançado novamente, e tem a oportunidade de atualizar a população ideal.

Outro trabalho apresentado é o de Ramezani (2014) que apontou a dificuldade em utilizar recursos de forma adequada e distribuída na computação em nuvem. Muitas requisições de máquinas virtuais com vários *data centers* e vários *hosts.* Neste sentido surge a necessidade de ter-se algoritmos eficazes que, baseados em fatores como CPU, memória, disco e largura de banda, busque e publique uma máquina virtual no local mais adequado.

Em Ramezani (2014), foi desenvolvido um algoritmo baseado em *particle swarm* com a finalidade de escolher o melhor local para escalonar uma máquina virtual nova ou alguma já publicada, visando o balanceamento de recursos.

A divisão foi realizada em duas categorias: computação intensiva e dados intensivos. Na computação intensiva são consideradas aplicações com alto custo computacional, como por exemplo, meteorologia e aplicações científicas. Para estas aplicações, o algoritmo considera o número de CPUs disponíveis para a tarefa, visando escalonar tarefas intensivas em *hosts* com alta capacidade de processamento. Nos dois casos também são considerados para cálculo a memória e disco.

Dados intensivos são aplicações que manipulam grandes quantidades de dados. Neste caso o algoritmo utiliza a largura de banda para realizar o cálculo, visando diminuir atraso na movimentação de grandes quantidades de dados.

Para determinar se uma máquina está sobrecarregada é utilizada a seguinte fórmula: V M r w = V Mw − V Met

Vmw é a carga de trabalho na VM e VMet é o número de tarefas em execução na VM. A máquina está sobrecarregada e deverá ter suas tarefas migradas quando: **0 ≤ V M r w ≤ 1.**

Na Figura 16 são mostrados os passos do algoritmo.

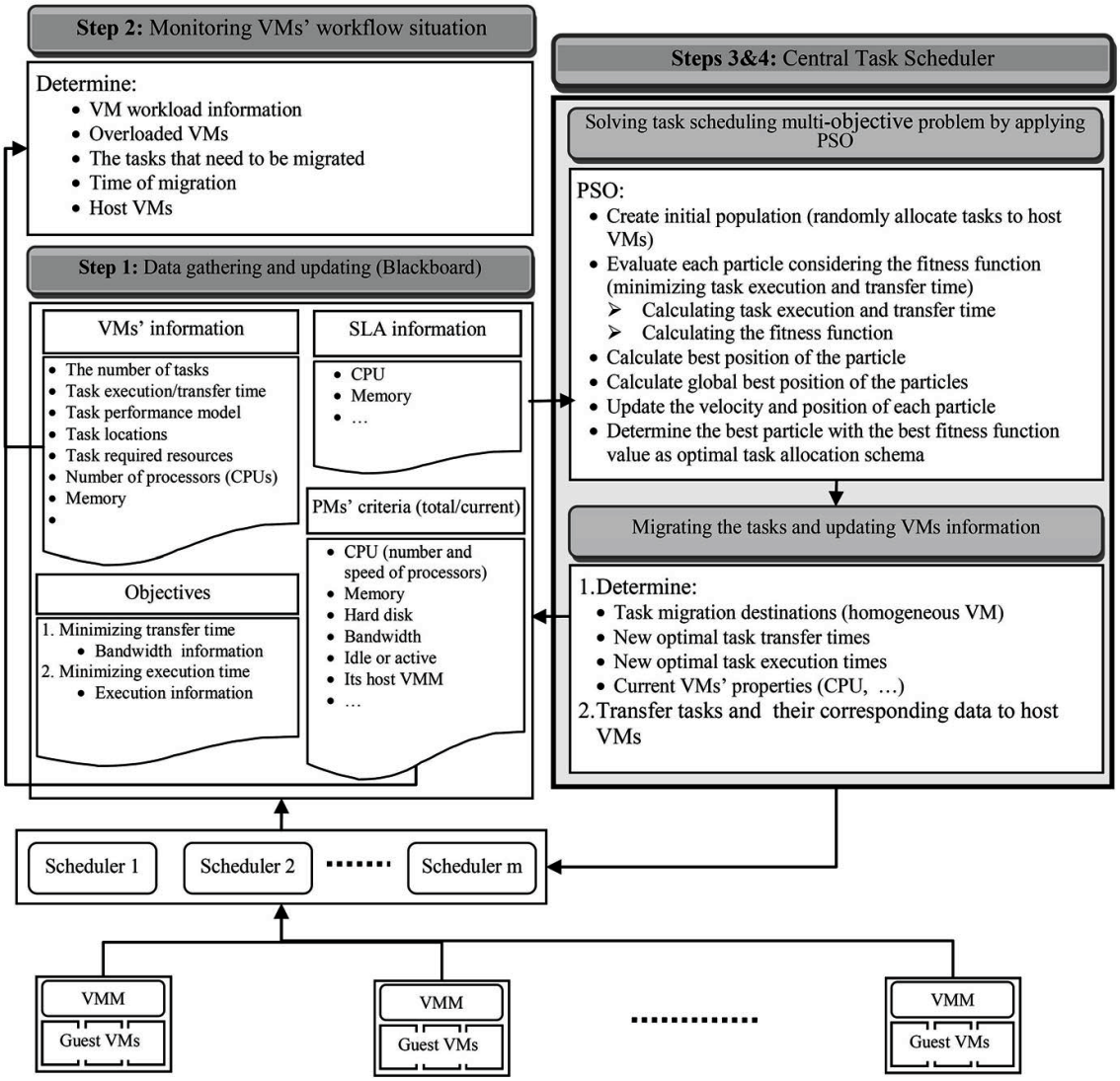

Figura 16 – Modelo conceitual (Ramezani 2014).

O algoritmo é descrito em 6 passos:

Passo 1 Busca de informações sobres Vms;

Passo 2 Monitorar informações sobre a carga de trabalho das Vms, determinar se a VM está sobrecarregada, determinar se deve ser migrada e o tempo de migração.

Passo 3 Busca o novo *host* para migrar a tarefa.

Passo 4 Com base nas informações obtidas das tarefas que estão sobrecarregando a VM e nas informações do novo *host,* calcula tempo de execução.

Passo 5 Transfere a tarefa para o novo *host* escolhido.

Passo 6 Atualiza informações iniciais de acordo com a nova configuração.

3.3.5 *Match Making*

O *Match making* foi criado com o objetivo de prover uma infraestrutura para facilitar a criação dos algoritmos, do ponto de vista do criador do algoritmo, e diminuir a complexidade de uma operação de escalonamento (PRAJAPATI, 2013).

A ferramenta deve armazenar a proveniência de cada processo de alinhamento ocorrido. Ou seja, deve acomodar um histórico de todos os alinhamentos, contendo os parâmetros de execução, etapas do algoritmo que foram processadas, funções de similaridades utilizadas e representações de dados aplicadas para que o processo fosse concluído. Esses dados sobre proveniência precisam estar disponíveis para o usuário poder comparar os diferentes resultados obtidos.

Por fim, com o intuito de fomentar a comparação de resultados, o *MatchMaking* deve ser capaz de realizar operações de alinhamento em série, automaticamente. Ou seja, o usuário poderá definir esquemas de origem e de destino (e seus *datasets*) e especificar uma série de operações de alinhamento que devem ser executadas. Após isso, a infraestrutura executará cada operação de alinhamento especificada a série, de forma automática, sem necessidade de interação adicional com o usuário.

A arquitetura da infraestrutura é descrita através do modelo conceitual de seu banco de dados interno. Um esquema (*Schema*) é definido como uma agregação de elementos: classes, propriedades e instâncias.

Uma propriedade contextualizada define um relacionamento entre classes e propriedades, capturando o fato de que uma mesma propriedade pode estar definida para mais de uma classe do esquema. Em cada esquema, podemos ter vários conjuntos de dados *(datasets)* que representam instâncias de classes obtidas em um determinado momento. O *Matchmaking* consegue controlar vários desses conjuntos de forma independente. Nas técnicas de

alinhamento extensivas ou híbridas, os valores das propriedades de cada uma das instâncias de classe podem alterar o resultado do alinhamento. Portanto, é importante diferenciar que conjunto de dados foi utilizado em um alinhamento.

Um algoritmo de alinhamento (*Matcher*) possui uma lista de parâmetros que deve, obrigatoriamente, ser preenchida no momento da sua execução. Ele também pode aplicar funções de similaridade em representações dos dados do esquema. O algoritmo pode ser dividido em vários passos (também chamados de subalgoritmos), cada qual considerado como um novo procedimento com parâmetros e aplicações de funções de similaridade próprias. Com essa arquitetura, é possível até mesmo criar um procedimento que agrega outros algoritmos independentes e calcula a similaridade de dois esquemas através do resultado obtido pela execução dos outros. Por exemplo, com o resultado obtido entre dois algoritmos de alinhamento, podemos aplicar uma equação que calcula o maior, menor ou média dos dois valores e considerar esse o grau de similaridade.

Em Prajapati et al(2013), foi apresentado o algoritmo *match making* como escalonador de máquinas virtuais para computação em nuvem. No primeiro o algoritmo filtra os nós ou hosts que não atendem aos requisitos de VM e não tem recursos suficientes (como CPU, memória, Processadores etc) para alocar e executar a VM. A classificação será dada para os nós conforme as informações coletadas pelo monitoramento. Se alguma variável entrar em monitoramento, será incluído na expressão de classificação. O resultado do *ranking* da expressão é dada ao planejador da nuvem e ao monitoramento. O algoritmo então toma a decisão de alocação de VMs e de reconfiguração. O objetivo do algoritmo é priorizar os recursos que são mais adequados para a VM. Os recursos com maior classificação são usados primeiro para alocar as VMs.

3.3.6 *Memory Aware*

Ainda em Prajapati et al(2013), foi apresentado o algoritmo *Memory Aware* para escalonar recursos em computação em nuvem.

No *Memory Aware*, o agendador coleta informações sobre o comportamento do cache de cada VM de cada máquina física e migra VMs que podem potencialmente reduzir a falta de cache geral e latências médias de acesso à memória no sistema da nuvem. Em cada verificação do monitor (Cache do último nível) As falhas do cache LLC / L3 com contadores de monitoramento de desempenho de hardware e envia periodicamente a credibilidade por VM e a afinidade NUMA com informações para o planejador da nuvem. Com base no status da VM e informações de todos os nós, o planejador da nuvem faz decisões globais de agendamento. O Agendador *Cache-Aware* que trabalha em duas fases locais (agendamento baseado no comportamento do cache de um nó) e global (agendamento baseado no comportamento do cache de todos os nós) e NUMA-*Aware scheduler* que trabalham em fase global apenas. Esses agendadores conscientes de memória reduzem as contenções de memória entre VMs do mesmo *host*, mas o número de migrações aumentará.

Em processos de um único thread, e um sistema de tempo compartilhado, onde os processos correm por tempo especificado. Além disso, todo o contexto do processador se altera ao mesmo tempo que seria feito na programação. Esses pressupostos não são fundamentais para a abordagem, uma vez que existem processadores e um máximo de processos que podem ser executado ao mesmo tempo.

Para agendar mais processos, o sistema é de tempo compartilhado. Se os processos tiverem grande requisitos de memória, considera-se compartilhamento de tempo, com tempo maior para amortizar o tempo de mudança de contexto. Alternativamente, é possível considerar um sistema de lote onde os processos são executados até a conclusão. Como isso exigirá

novas atribuições de agendamento assume-se um sistema de tempo compartilhado para simplificar a exposição.

3.3.7 *Trust Aware Distributed and Collaborative Scheduler(TADCS)*

Trust Aware Distributed and Collaborative Scheduler (TADCS) apresenta um agendador para colocação de VM e provisionamento em nuvem. Baseia-se na arquitetura ponto a ponto, permitindo um modelo completamente centralizado. A programação deste algoritmo toma decisões com base em recursos dinâmicos que calculam uma pontuação e nos objetivos do usuário (isto é, unidades de processamento, memória, espaço em disco e largura de banda). A pontuação é avaliada com base em avaliação estática e uso de recursos dinâmicos, mas também usa a cota de recursos associado a cada vm. As cotas de recursos são de dois tipos:

(1) *Soft Quota*: Quantidade de recursos dedicados à VM;

(2) *Hard Quota*: Quantidade de recursos usados pela VM isso dedicado a nenhuma VM em particular.

Os objetivos do usuário facilitam a expressão de qualidade de confiança e proteção e a implantação da arquitetura HPC. Esses objetivos são dados pelo usuário para cada uma de suas VMs. Este algoritmo é um algoritmo perfeito para provisionamento seguro de recursos que separam as máquinas virtuais da mesma máquina física e migram para o outra máquina física ou *data center*. É necessário pensar sobre o provisionamento seguro de recurso que não afete o QoS que vem com o Acordo de SLA. Para dar uma solução segura com os vizinhos das VMs faz-se necessárias funções de pontuação adequadas que podem ser calculadas com base em quantos tempos a VM é migrada por causa de uso excessivo dos recursos de outras máquinas virtuais e pode ser otimizado pelo uso do algoritmo *Trust Aware Scheduler* existente. (CORNABAS, 2011).

Capítulo 4 Implementação e Testes

Este trabalho utiliza o algoritmo *particle swarm* como base para escalonar máquinas virtuais na classe de infraestrutura como serviço(IaaS). Busca o ambiente que atenda a necessidade de recursos solicitados e o QoS contratado. Os testes foram realizados no simulador iSPD (MENEZES et al., 2012), disponível para download em https://www.researchgate.net/publication/271524398_iSPD_v20__Software_Download, em uma máquina com sistema operacional Windows 10 64 bits com 8 GB de memória RAM, processador i5 de 3.20 GHz e disco SSD de 400 GB.

O software iSPD simula um ambiente real de computação em nuvem, nele é possível selecionar qual tipo de ambiente deseja simular: *grid, IaaS* (infraestrutura como serviço*) ou PaaS* (plataforma com serviço). Após selecionado o tipo de ambiente, monta-se a estrutura com máquinas físicas e/ou *cluters,* cada um com suas respectivas configurações de memória, disco, processador.

Também é possível fazer as conexões de rede e determinar qual a largura de banda, fator de carga e latência. Cria-se as máquinas virtuais com suas respectivas configurações e vinculadas a um servidor físico.

Para a simulação do ambiente em funcionamento, também seleciona-se qual o nível de carga em cada máquina física e em cada máquina virtual e qual o algoritmo será usado para fazer o escalonamento.

No simulador iSPD foi implementada a classe SimulacaoParticleSwarm, esta classe é responsável por ler os dados dos ambientes simulados, executar o algoritmo *particle swarm* e salvar o resultado no banco de dados mysql 5.7.21. O método BuscaDados é o responsável por ler os dados do ambiente criado na simulação, faz a verificação de quais máquinas atendem a demanda e marca com 1 e armazena os valores nas variáveis.

O método GeraPartículas é o responsável por inicializar as partículas de memória, disco e processadores com valores aleatórios que são gerados pela classe *random* do java. Entre os valores criados faz as verificações e escolhe qual será iniciado como melhor posição global.

A cada iteração, o método AtualizaParticula altera a direção de cada partícula conforme o melhor valor local e melhor valor global encontrado. O método Resultado, salva no banco de dados os valores encontrados.

Segue explicação dos atributos criados:

- ValorReferencial - É o valor tomado como referência para verificar se o algoritmo alcançou valor próximo ao desejado;
- Processador – É quantidade de processadores em *giga-hertz,* que foram solicitados ao algoritmo para a máquina virtual solicitada;
- Memória - É quantidade de memória *ram* em *megabyte,* que foi solicitada ao algoritmo para a máquina virtual solicitada;
- Disco - É o tamanho do disco *gigabyte,* que foi solicitado ao algoritmo para a máquina virtual solicitada;
- QuantidadeParticulas – É a quantidade de partículas que serão iniciadas para busca da solução;
- Peso_local – Este atributo representa qual o peso de cada partícula na verificação do algoritmo para encontrar a melhor solução;
- Peso_global – Este atributo representa quanto o peso global terá influência no cálculo do algoritmo em busca da solução;
- Iteracoes – Representa quantas vezes o algoritmo será repetido em busca da solução;
- Gbest_memoria – Qual o melhor valor global encontrado para memória.;
- Gbest_procs – Representa o melhor valor global encontrado para processadores;
- Gbest_disco – Representa o melhor valor global encontrado para disco.

Segue explicação dos métodos existentes na classe jPrincipal:

- EscolherClasse – Qual tipo de classe será testada no ambiente de nuvem – Infraestrutura como serviço, *software* como serviço ou plataforma como serviço;
- ConfigurarVMs – Utilizado para setar a configuração das máquinas virtuais;
- TipoModelo – Qual modelo será utilizado.

Segue explicação dos atributos existentes na classe jPrincipal:

- jMenuItemGerenciarActionPerformed – Gerenciar o ambiente de máquinas virtuais criado;
- jMenuItemSobreActionPerformed – Breve explicação sobre o sistema e versões;
- jMenuItemInglesActionPerformed – Utilizado para configura o *software* quando o idioma inglês for selecionado;
- jMenuItemPortuguesActionPerformed - Utilizado para configura o *software* quando o idioma prtuguês for selecionado;
- jToggleButtonMaquinaActionPerformed – Utilizado para criar máquina virtual;
- jToggleButtonRedeActionPerformed – Serve para selecionar os nós de rede que farão a conexão entre as máquinas;
- jToggleButtonClusterActionPerformed – Utilizado para criar *cluster* de máquinas virtuais;
- jToggleButtonInternetActionPerformed – Para criar conexões de internet entre os *clusters*;
- jButtonTarefasActionPerformed – Para selecionar as tarefas que serão executadas durante a simulação;
- jButtonSimularActionPerformed – Utilizado para iniciar a simulação do ambiente;
- jMenuItemNovoActionPerformed – Criar um novo ambiente de nuvem.

No diagrama 1 é apresentada a classe criada e seu relacionamento com a classe existente no iSPD.

Na classe jPrincipal (classe existente no simulador iSPD) na ação do botão jButtonSimularActionPerformed, foi instanciada a classe SimulacaoParticleSwarm e invocado o método BuscaDados.

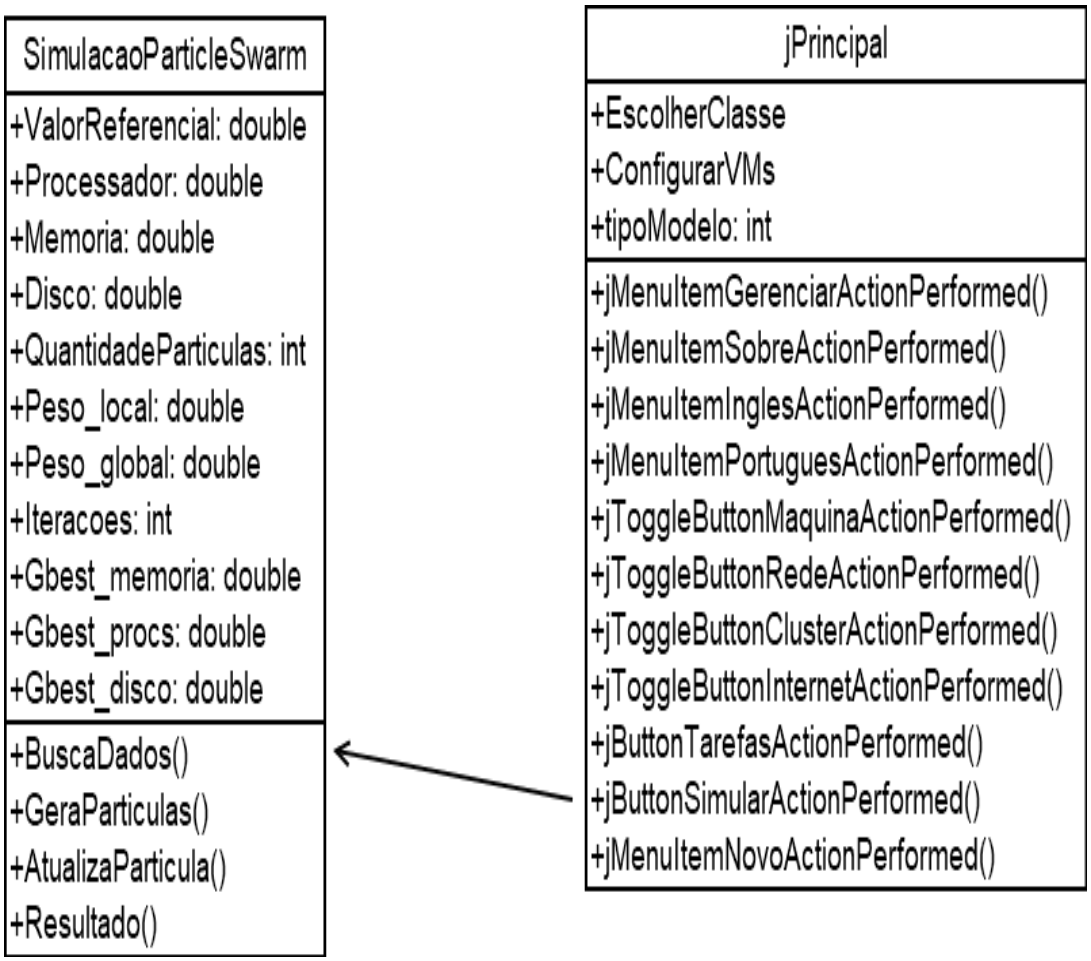

Diagrama 1 – Diagrama de Classes

No diagrama 2 é mostrada a estrutura de criação das tabelas que foram usadas para armazenar os dados coletados e usá-los na verificação do algoritmo. A tabela MAQUINA armazena os valores do recursos (memória, disco, processador e frequência do processador) de cada máquina criada no simulador, tem também um campo bit que é utilizado para marcar as máquinas que atendem a solicitação inicial.

A tabela RECURSOS_DC armazena a soma de recursos de todas as máquinas que fazem parte de um *data center* e também qual o número do teste realizado.

Na tabela de RESULTADO são salvos os valores referentes a cada partícula em cada iteração. REF_MEM, REF_DISCO e REF_PROCS, são os valores de referência (este valor é o valor máximo disponível em um *data center*) para aquele teste. QTD_PARTICULA armazena quantas partículas foram iniciadas naquele teste, NUM_PARTICULA identifica de qual partícula é o teste registrado. PBEST_MEM, PBEST_DISCO e PBEST_PROCS são os melhores valores encontrados para aquela partícula até aquel momento. GBEST_MEM, GBEST_PROCS E GBEST_DISCO são os melhores valores encontrados para o grupo de partículas até aquele momento. MEM_SOLICITADA, DISCO_SOLICITADO E PROCS_SOLICITADO são os valores de recursos que foram inicialmente solicitados para disponibilizar para crtiação da máquina virtual. QTD_DATACENTER é a quantidade de *data centers* que estão sendo analisados naquele teste.

MEM_SOLICITADA, DISCO_SOLICITADO, PROCS_SOLICITADO, são respectivamente, a quantidade de memória, disco e processadores que foram solicitados ao algoritmo.

GBEST_MEM, GBEST_DISCO E GBEST_PROCS são os melhores valores encontrados para memória, disco e processadores durante uma determinada verificação.

Na tabela RECUROS_DC, são armazenados os valores disponíveis em cada *data center* . A coluna ID, é o identificador único do *data center,* MEMORIA_TOTAL, DISCO_TOTAL, E PROCS_TOTAL, são respectivamente a soma de todos recursos (memória, disco e processadores) disponíveis naquele *data center,* NUM_TESTE é o identificador do teste que foi feito para aqueles recursos disponíveis.

Na tabela MAQUINA, tem a coluna ID_MAQUINA que é o identificador único de máquina, NOME é o nome que foi dado para a máquina no momento da criação no simulador, CPUS é a quantidade de processadores virtuais que aquela máquina possuí, FREQUENCIA_PROC é a frequência dos

processadores em GHz, DISCO é o tamanho do disco em Gb que aquela máquina possui, a coluna ATENDE é utilizada para marcar a máquina com 1 caso ela tenha o mínimo de recursos para atender a demanda solicitada, se não tiver recursos suficientes, ela será marcada com zero e não será considerada na soma de recursos do *data center* no momento da verificação do algoritmo.

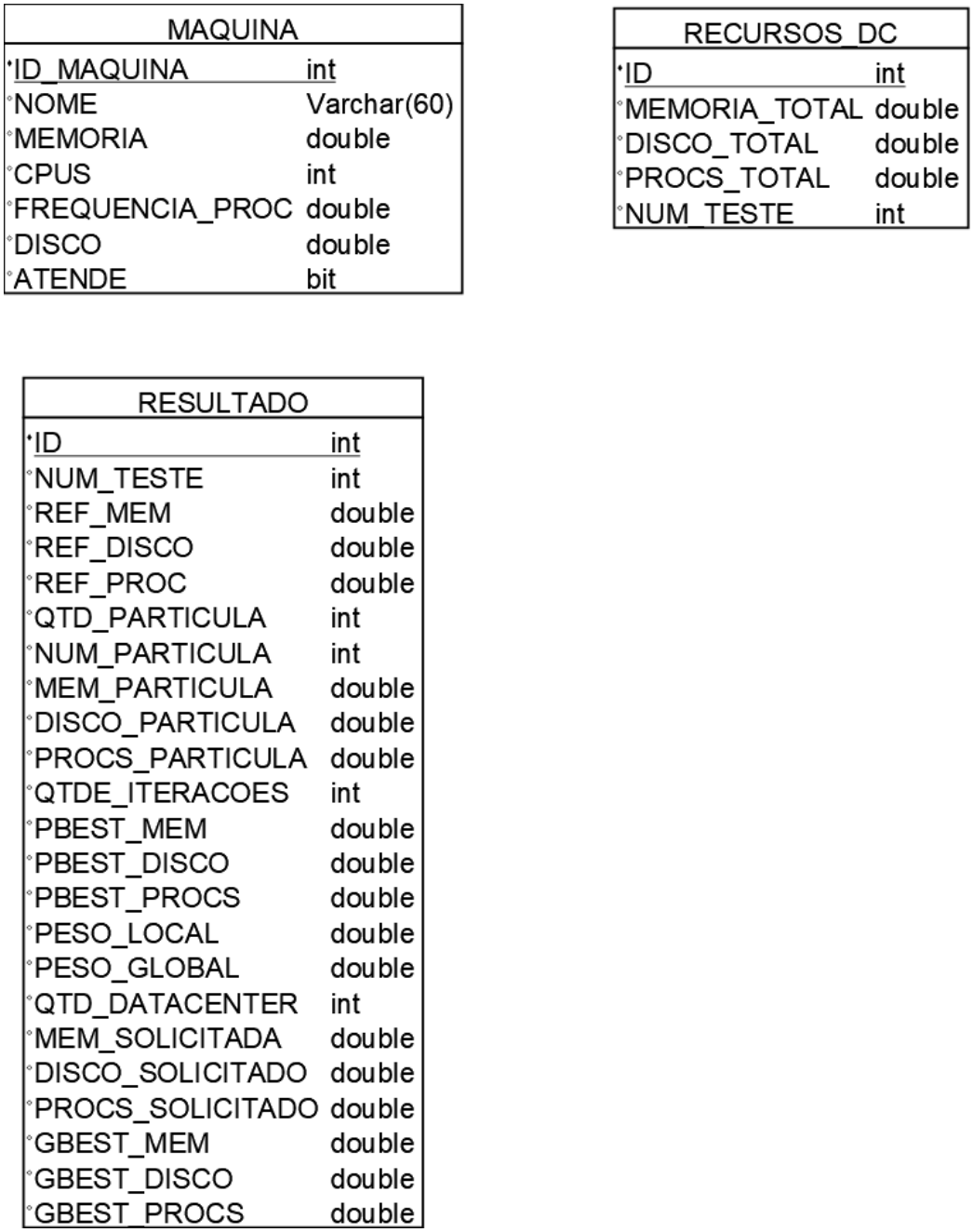

Diagrama 2 – Modelo físico do banco de dados

Na Figura 17 é mostrado o ambiente criado com 5 *data centers* com 21 máquinas físicas no simulador com a seguinte composição:

- Três máquinas físicas - dois *data centers;*
- Quatro máquinas físicas - um *data center;*
- Cinco máquinas físicas - um *data center;*
- Seis máquinas físicas – um *data center.*

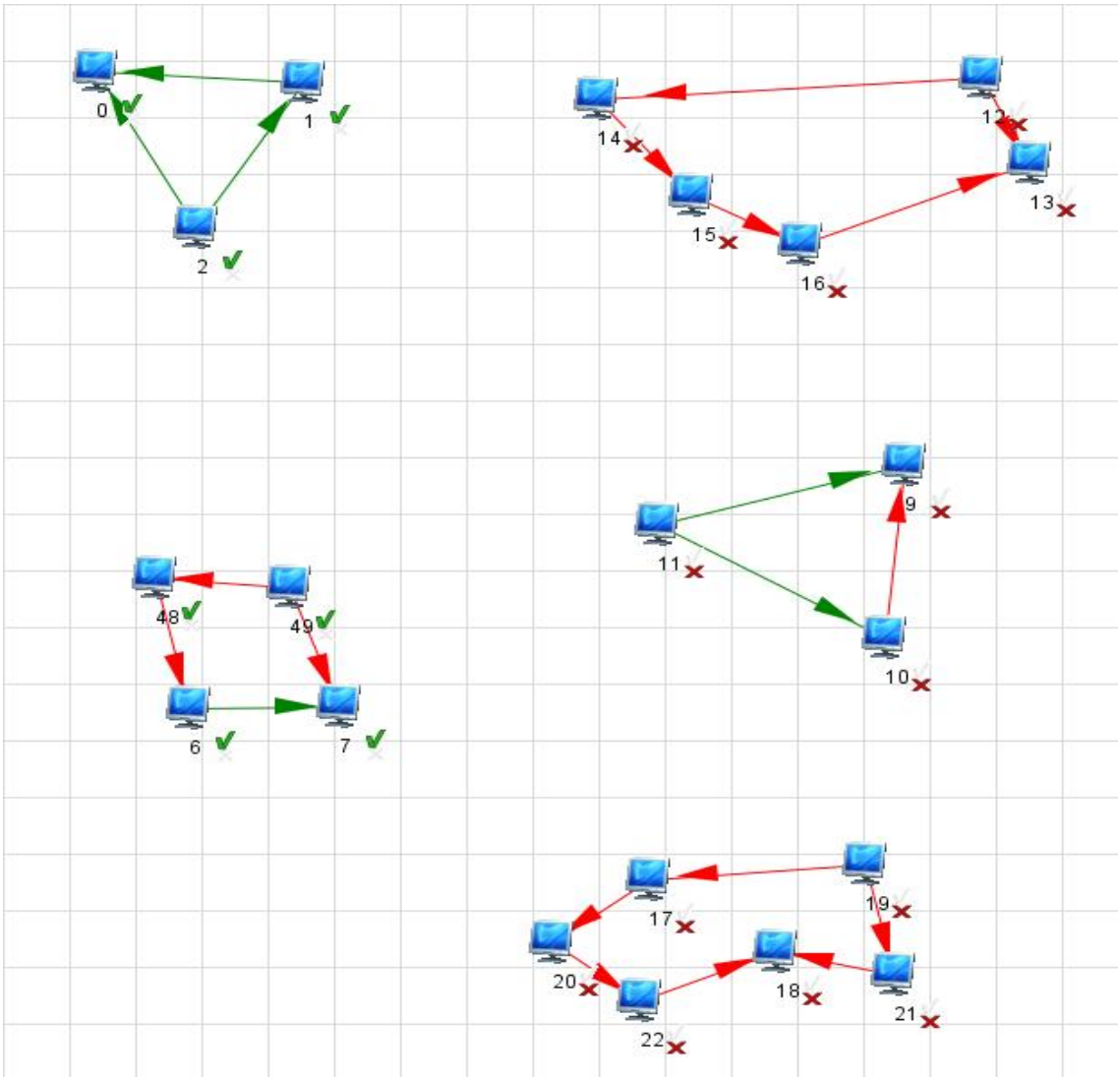

Figura 17 – Ambiente 1 com 5 *data centers* e 21 máquinas físicas.

Na Figura 18 é mostrado o ambiente criado com 10 *data centers* com 71 máquinas físicas no simulador com a seguinte composição:

- Três máquinas físicas - um *data center;*
- Quatro máquinas físicas - um *data center;*

- Cinco máquinas físicas - um *data center;*
- Seis máquinas físicas – três *data centers.*
- Sete máquinas físicas - um *data center;*
- Oito máquinas físicas - um *data center;*
- Dez máquinas físicas - um *data center;*
- Dezesseis máquinas físicas – *um data center.*

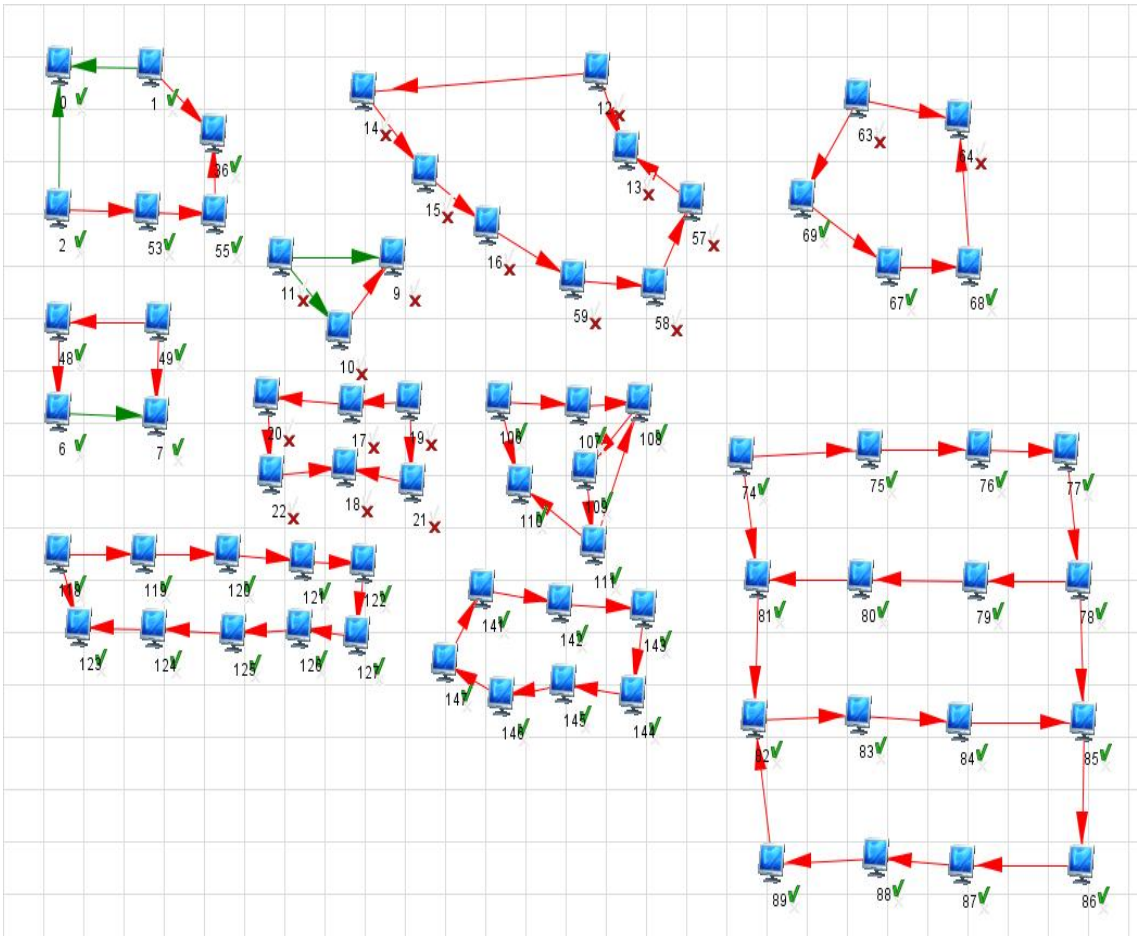

Figura 18 – Ambiente 2 com 10 *data centers* e 71 máquinas físicas.

Na Figura 19 é apresentado o ambiente criado com 30 *data centers* com 200 máquinas físicas no simulador.

- Três máquinas físicas - três *data centers;*
- Quatro máquinas físicas - dois *data centers;*
- Cinco máquinas físicas - quatro *data centers;*
- Seis máquinas físicas – nove *data centers.*
- Sete máquinas físicas - quatro *data centers;*

- Oito máquinas físicas – três *data centers;*
- Nove máquinas físicas - um *data center;*
- Dez máquinas físicas – dois *data centers;*
- Doze máquinas físicas – um *data center;*
- Dezesseis máquinas físicas - um *data center.*

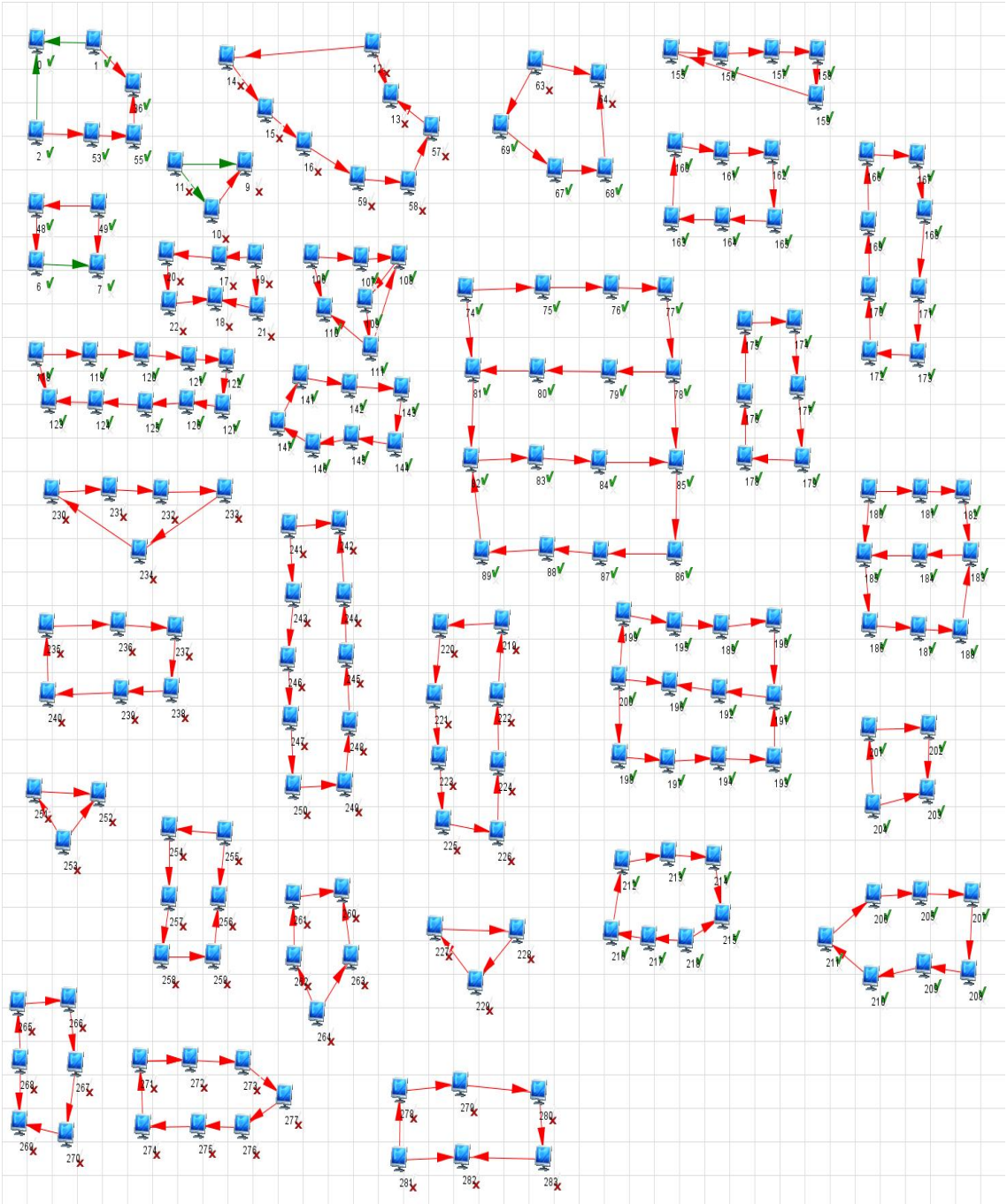

Figura 19 – Ambiente 3 com 30 *data centers* e 200 máquinas físicas.

Antes do algoritmo analisar em qual *data center* a máquina virtual será alocada, o módulo de monitoramento de recursos faz a busca em todos *data centers* para verificar se cada um tem pelo menos uma máquina física para atender a demanda solicitada. Durante a verificação, marca com 0 as máquinas que não têm recursos suficientes para atender a requisição de recursos solicitada e com 1 cada máquina com condições de atender a demanda apresentada. Apenas as máquinas marcadas com 1 comporão o total de recursos disponíveis no *data center.* Para manter o balanceamento de carga, o cálculo deverá considerar a razão do total de recursos do *data center* pelo total de recursos disponíveis considerando apenas recursos de máquinas físicas que anteriormente foram marcadas com 1. O algoritmo utiliza para cálculo os recursos disponíveis de *CPU,* disco e memória, dentro do ambiente de nuvem.

Inicia-se o algoritmo com um conjunto de partículas. Cada uma segue um caminho em direção a solução mais adequada. O algoritmo calcula, baseado no melhor resultado global e no melhor resultado local, o caminho em direção ao melhor *data center,* o caminho das requisições são alteradas ao longo do tempo durante a execução do algoritmo até que chegue à solução mais adequada.

Foi realizado um teste inicial com 5 *data centers* com 21 máquinas físicas para verificar a viabilidade do algoritmo, este teste foi realizado para verificar se o algoritmo alcançava o valor esperado, os resultados alcançados são demonstrados a seguir.

O *data center* com mais recursos disponíveis tinha em seu total livre, já considerando apenas as máquinas que tinham condições de atender a requisição solicitada era de 312.000 MB de memória RAM, 31.000 GB de espaço em disco e 128,8 GHz de processamento.

Após todas verificações, a configuração na qual o algoritmo mais se aproximou do resultado ideal, que é o *data center* com mais recursos disponíveis no momento, foi com 500 partículas, 1000 iterações e com peso global em 0,5.

Capítulo 5 Resultados

Os valores iniciais dos testes foram baseados nos trabalhos apresentados na seção 3. Durante os testes, a constante de peso da melhor posição global foi alterada, inicialmente em 0,5 e depois em 0,7 e os seus resultados foram comparados para que se chegue a melhor configuração do algoritmo para resolver o problema proposto. Os testes foram realizados com 5, 100, 500 e 1000 partículas, variando em todos os testes a quantidade de iterações em 50, 100, 500 e 1000 vezes, no total foram gerados 38723 registros para serem analisados.

Para todos os testes foi enviado ao simulador a alocação de uma máquina com 2 processadores de 2.8 GHz, memória de 8 GB e disco de 200 GB. Os resultados estão agrupados pela quantidade de data centers. Foram criados 3 ambientes no simulador, sendo o primeiro com 5 *data centers* com 21 máquinas físicas, o segundo com 10 data centers com 71 máquinas físicas e o terceiro com 30 data centers com 200 máquinas físicas.

Para os três ambientes foi possível observar que quanto maior a quantidade de partículas os resultados foram mais próximos dos valores que foram tomados como referência (*data center* com mais recursos disponíveis no momento da verificação). Também foi constatado que quanto mais iterações e maior a quantidade de partículas, os resultados tiveram a menor variação de valores.

O algoritmo tem duas possibilidades de parada: pela quantidade de iterações ou se atingir o máximo de recursos disponíveis. Em todos os testes, a parada foi a quantidade de iterações, o algoritmo não chegou a atingir a quantidade máxima de recursos disponíveis.

A seguir, serão apresentados os detalhes dos testes realizados.

5 *data centers* com 21 máquinas físicas:

O data center com mais recursos disponíveis tinha em seu total livre, já considerando apenas as máquinas que tinham condições de atender a requisição solicitada era de 312.000 MB de memória RAM, 31.000 GB de espaço em disco e 128,8 GHz de processamento.

Após todas verificações conforme capítulo anterior, a configuração na qual o algoritmo mais se aproximou do resultado ideal, que é o *data center* com mais recursos disponíveis no momento, foi com 500 partículas, 1000 iterações e com peso global em 0,5.

O primeiro teste foi realizado com os parâmetros utilizados por Yuan (2014) e Ramezani (2014) com o intuito de validar os resultados, na Tabela 1 é possível observar que os valores alcançados com 6 partículas, que foi o utilizado no trabalho de Yuan (2014), com peso local e global em 0,5 com 293 iterações foi o que mais se aproximou do valor desejado apresentado na célula memória, disco e processadores.

Ainda na tabela 1, observando os resultados utilizando os parâmetros de Ramezani (2014), o resultado também foi satisfatório utilizando 20 partículas, peso local e global 0,5 com 2000 itrerações.

	REFERÊNCIA					
	MEMÓRIA	DISCO	PROCESSADORES			
	312000	31000	128.8			
QTD_PARTICULA	GBEST_MEM	GBEST_DISCO	GBEST_PROCS	ITERACOES	PESO_LOCAL	PESO_GLOBAL
6	163145.00	28727.00	113.60	200	0.50	0.50
6	295450.00	25914.00	124.60	293	0.50	0.50
6	264121.00	28693.00	105.60	500	0.50	0.50
Yuan(2014)						
20	255499.00	30132.00	120.60	2000	0.50	0.50
Ramezani(2014)						

Tabela 1: Resultados iniciais.

A Tabela 2 apresenta os resultados obtidos para 5 data centers com 5 partículas, mantendo o peso local em 0,5 e variando o peso global em 0,5 e 0,7.

É possível observar que os resultados variaram bastante, ficando mais próximo do resultado ideal apresentado na célula memória, disco e

processadores, foi com 1000 partículas porém com as constantes de peso local e peso global com valores iguais, ou seja, 0,5.

	REFERÊNCIA					
	MEMÓRIA	DISCO	PROCESSADORES			
	312000	31000	128.8			
QTD_PARTICULA	GBEST_MEM	GBEST_DISCO	GBEST_PROCS	ITERACOES	PESO_LOCAL	PESO_GLOBAL
5	227601.00	26381.00	124.60	50	0.50	0.70
5	294319.00	18862.00	119.60	50	0.50	0.50
5	246830.00	29379.00	125.60	100	0.50	0.70
5	283017.00	28139.00	117.60	100	0.50	0.50
5	232431.00	28276.00	95.60	500	0.50	0.70
5	292369.00	19428.00	123.60	500	0.50	0.50
5	273558.00	19543.00	111.60	1000	0.50	0.70
5	300831.00	30080.00	119.60	1000	0.50	0.50

Tabela 2: Resultados para 5 *data centers* com 5 partículas.

Na Tabela 3 são apresentados os resultados com 10 partículas, neste teste é possível observar que houve maior variação nos resultados, o melhor resultado para gbest_mem, foi 1000 iterações com peso local e global em 0,5, já para gbest_disco foi com 50 iterações com peso local em 0,5 e peso global em 0,7. Já para gbest_procs o melhor resultado foi com 100 e com 1000 iterações com peso local em 0,5 e peso global em 0,7.

	REFERÊNCIA					
	MEMÓRIA	DISCO	PROCESSADORES			
	312000	31000	128.8			
QTD_PARTICULA	GBEST_MEM	GBEST_DISCO	GBEST_PROCS	ITERACOES	PESO_LOCAL	PESO_GLOBAL
10	239708.00	30099.00	83.60	50	0.50	0.70
10	273873.00	29109.00	111.60	50	0.50	0.50
10	287730.00	27909.00	117.60	100	0.50	0.50
10	300868.00	29992.00	126.60	100	0.50	0.70
10	271620.00	28579.00	110.60	500	0.50	0.70
10	285944.00	29658.00	120.60	500	0.50	0.50
10	304154.00	27787.00	126.60	1000	0.50	0.70
10	311452.00	28841.00	115.60	1000	0.50	0.50

Tabela 3: Resultados para 5 *data centers* com 10 partículas.

10 *data centers* com 71 máquinas físicas:

O *data center* com mais recursos disponíveis tinha em seu total livre, já considerando apenas as máquinas que tinham condições de atender a requisição solicitada era de 640.000 MB de memória RAM, 55.550 GB de espaço em disco e 647,20 GHz de processamento.

Após todas verificações conforme seção 4, a configuração na qual o algoritmo mais se aproximou do resultado ideal, que é o *data center* com mais recursos disponíveis no momento, foi com 1000 partículas, 1000 iterações e com peso global em 0,5 conforme apresentado na Tabela 4, as células memória, disco e processadores mostram os valores ideais para o algoritmo alcançar.

	REFERÊNCIA					
	MEMÓRIA	DISCO	PROCESSADORES			
	640000	55550	647.2			
QTD_PARTICULA	GBEST_MEM	GBEST_DISCO	GBEST_PROCS	ITERACOES	PESO_LOCAL	PESO_GLOBAL
1000	638855.00	55457.00	646.60	50	0.50	0.50
1000	639680.00	55541.00	646.60	100	0.50	0.50
1000	639603.00	55466.00	646.60	500	0.50	0.50
1000	639697.00	55507.00	646.60	1000	0.50	0.50
1000	638257.00	55545.00	646.60	50	0.50	0.70
1000	639795.00	55426.00	645.60	100	0.50	0.70
1000	636312.00	55531.00	646.60	500	0.50	0.70
1000	639793.00	55514.00	646.60	1000	0.50	0.70

Tabela 4: Resultados para 10 *data centers* com 1000 partículas.

30 data centers com 200 máquinas físicas:

O *data center* com mais recursos disponíveis tinha em seu total livre, já considerando apenas as máquinas que tinham condições de atender a requisição solicitada era de 3.672.000 MB de memória RAM, 399.820 GB de espaço em disco e 4.353,80 GHz de processamento.

Foi realizado teste com peso local 0,5 e peso global 0,3 para verificar se ficaria preso ao mínimo local, o resultado foi 3.667.333 MB para memória,

4.349.60 GHz para processadores e 399.738 GB para disco. Também foi realizado teste com peso local 0,7 e peso global 0,3, o resultado foi 3.671.321 MB para memória, 4.350,60 GHz para processamento e 399.609 GB para disco. Mesmo com peso local maior que o peso global, o algoritmo não ficou preso ao mínimo local.

Após a verificação de todos os parâmetros e testes de acordo com capítulo anterior, foi possível observar que com 1000 partículas, os valores dos resultados do algoritmo foram os mais próximos do *data center* com mais recursos disponíveis. A Tabela 3 ilustra os resultados encontrados para 1000 partículas, com 50, 100, 500 e 1000 iterações. É possível observar que os valores encontrados com 500 e 1000 iterações foram ao que mais se aproximaram dos valores ideais, que são apresentados nas células memória, disco e processadores. Mesmo alterando a constante de peso global para 0,7, não houve alteração significativa nos resultados, isso mostra que os resultados não ficam presos no mínimo local mesmo quando os valores das constantes de peso local e peso global ficam com os valores iguais, o que nestes teste foi o valor de 0,5.

	REFERÊNCIA					
	MEMÓRIA	DISCO	PROCESSADORES			
	3672000	399820	4353.80			
QTD_PARTICULA	GBEST_MEM	GBEST_DISCO	GBEST_PROCS	ITERACOES	PESO_LOCAL	PESO_GLOBAL
1000	3671436.00	399482.00	4347.60	50	0.50	0.50
1000	3668826.00	399072.00	4351.60	100	0.50	0.50
1000	3670616.00	399255.00	4352.60	500	0.50	0.50
1000	3670907.00	399569.00	4345.60	1000	0.50	0.50
1000	3668995.00	399265.00	4344.60	50	0.50	0.70
1000	3670231.00	399615.00	4348.60	100	0.50	0.70
1000	3671794.00	398837.00	4338.60	500	0.50	0.70
1000	3664749.00	399580.00	4351.60	1000	0.50	0.70

Tabela 5: Resultados para 30 *data centers* com 1000 partículas. Fonte (Próprio autor).

No Gráfico 1 mostrada a variação dos resultados com peso local 0,5 e peso global 0,7 para memória em MB. O valor de referência, que é o *data center* com mais recursos disponíveis no momento da análise, é 3.672.000 MB de memória, 399.820 Gb de disco e 4353,80 GHz de processadores, é possível

verificar no gráfico que com peso global 0,7 e 500 iterações foi o ponto no qual o algoritmo alcançou o valor mais próximo do ideal. No entanto, é possível verificar que com peso global 0,5 a diferença foi bem pequena e também a variação dos valores conforme aumenta a quantidade de iterações é menor com peso global 0,5.

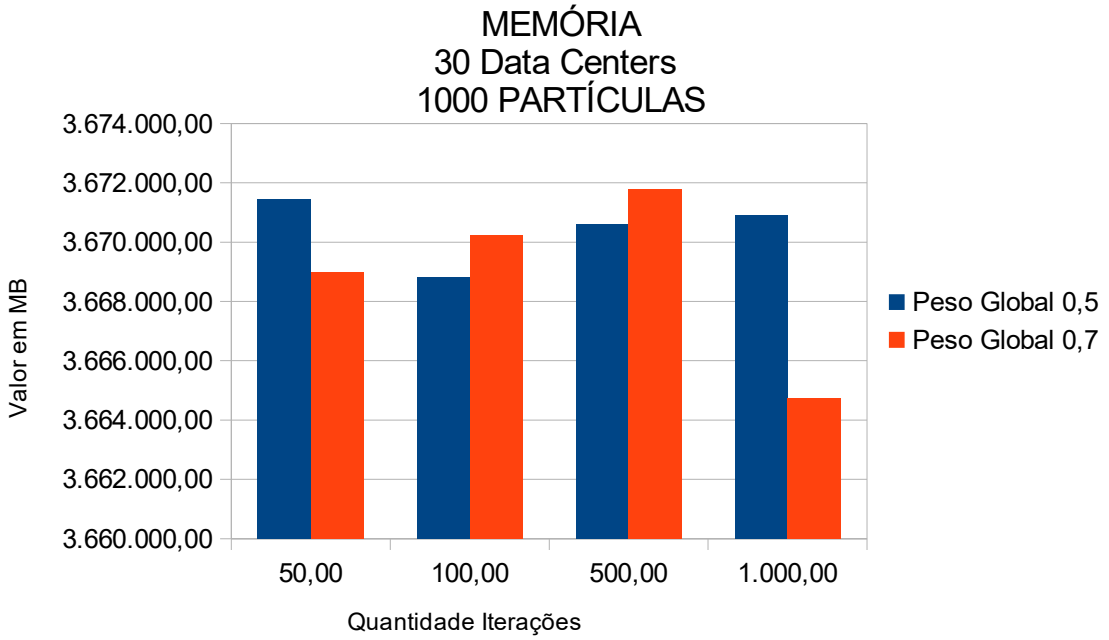

Gráfico 1: Memória. Fonte (Próprio Autor).

No Gráfico 2 é apresentada a variação dos resultados com peso local 0,5 e peso global 0,7 para disco em GB. Da mesma forma como aconteceu Com a memória, a configuração que mais se aproximou do melhor resultado foi com peso global em 0,7, porém com pouca diferença para o melhor valor encontrado com peso global em 0,5. E também com peso global em 0,5 a variação nos resultados foi menor.

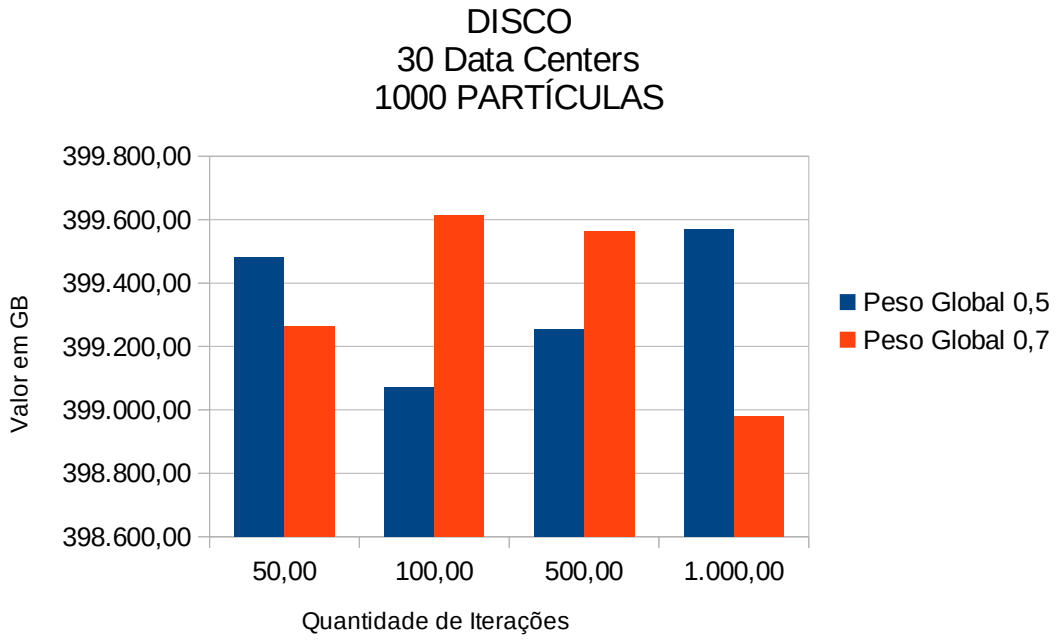

Gráfico 2: Disco. Fonte (Próprio Autor).

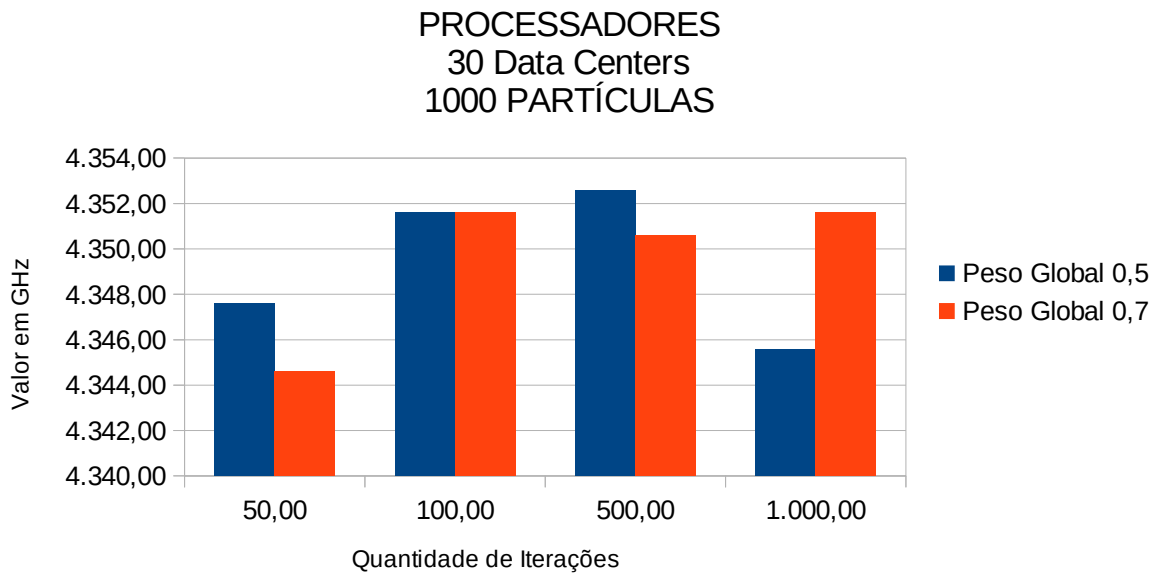

Gráfico 3: Processadores. Fonte (Próprio Autor).

No Gráfico 3 é mostrada a variação dos resultados com peso local 0,5 e peso global 0,7 para processadores em GHz. Neste caso o melhor resultado foi com peso global em 0,5 e com pouca diferença para o peso global 0,7. A

variação entre os resultados foi equilibrada comparando as duas configurações diferentes de peso global.

Na Tabela 6 é apresento o tempo total de execução do algoritmo para todos testes realizados com 30 *data centers,* é possível observar que a partir de 50 iterações e 500 partículas, o tempo de execução fica bem mais alto, também fica claro que o tempo de execução depende mais da quantidade de partículas do que da quantidade de iterações, pois com 50, 100, 500 ou 1000 iterações, o tempo de execução permanece entre 10000 e 10400 milisegundos.

	Tempo de Execução	
Iterações	**Partículas**	**Tempo em milisegundos**
50	5	215
50	100	1664
50	500	5391
50	1000	10289
100	5	172
100	100	1426
100	500	5399
100	1000	10197
500	5	218
500	100	1341
500	500	5413
500	1000	10383
1000	5	196
1000	100	1467
1000	500	5449
1000	1000	10246

Tabela 6: Tempo de execução. Fonte (Próprio autor)

Capítulo 6 Conclusão

Neste trabalho analisou-se o algoritmo *particle swarm* para verificar sua viabilidade no escalonamento de máquinas virtuais em um ambiente de computação em nuvem em infraestrutura como serviço, e encontrar a melhor configuração de parâmetros.

É possível concluir que, de acordo com os resultados apresentados, o algoritmo é viável para este tipo de escalonamento, além de que não demanda alto uso de processamento para os cálculos e o seu tempo de execução é da ordem de 10000 milisegundos. Porém quando o teste é realizado com poucas partículas há uma maior variação nos resultados encontrados e os melhores resultados foram com mais iterações e mais partículas, contudo o tempo de execução também aumenta consideravelmente como demonstrado na tabela 2.

Havia também a preocupação que as partículas ficassem presas ao mínimo local, por conta disto foram feitos testes com peso global (0,5) igual ao peso local (0,5) e peso global (0,7) maior que o peso local (0,5) e a constatação foi que, apesar dos melhores resultados encontrados terem sido com peso global 0,7, o algoritmo não ficou preso ao mínimo local, pois as variações de resultados foram pequenas.

Os testes mostraram também que quanto mais partículas são inicializadas e quanto mais iterações são realizadas, o algoritmo tende a se aproximar mais do valor ideal e a variação nos resultados são bem menores, no entanto, o tempo de execução aumenta consideravelmente.

Considerando os três ambientes criados no simulador (5 *data centers*, 10 *data centers* e 30 *data centers*) foi possível observar que o tamanho do ambiente não interfere na configuração dos parâmetros, pois as mesmas combinações trouxeram resultados parecidos. Desta forma o algoritmo pode ser utilizado para escalonar máquinas virtuais independente do tamanho do provedor de nuvem, ou até mesmo em um ambiente *intercloud*.

É possível concluir que, nos testes realizados e considerando todas variáveis, a melhor configuração do algoritmo é 1000 partículas, 500 iterações, peso local 0,5 e peso global 0,5.

Para trabalhos futuros pode-se trabalhar outras variações das constantes para verificar se é possível melhorar os resultados. Pode-se também trabalhar com outros recursos que fazem parte do ambiente de computação em nuvem, como por exemplo, largura de banda de rede, taxa máxima de transferência de dados em rede e latência de disco.

Referências Bibliográficas

ASSUNÇÃO , M. D.; COSTANZO , A.; BUYYA , R. computing to extend the capacity of clusters. A cost-benefit analysis of using cloud Cluster Computing, v. 13, p. 335–347, 2010. Disponível em http://dx.doi.org/10.1007/s10586-010-0131-x

ARMBRUST, M; FOX, A; GRITH, R; JOSEPH, A, D; KATZ, R; KONWINSK, A; LEE, G; A view of cloud computing. Communications of the ACM, 53(4),50-58. doi:10.1145/1721654.1721672, 2010.

BACHIEGA, N, G. Algoritmo de Escalonamento de Instância de Máquina Virtual na Computação em Nuvem. Universidade Estadual Paulista. São José do Rio Preto SP, 2014.

BARREIROS JUNIOR, W. O.. Escalonador de Tarefas para o Plataforma de Nuvens Federadas BioNimbuZ Usando Beam Search Iterativo Multiobjetivo. Tese de doutorado. Universidade de Brasília. Brasília DF Brasil, 2016.

BUYYA , R.; RANJAN , R.; CALHEIROS , R. N. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2010, Busan, South Korea, May 21-23), LNCS, Springer, Germany, 2010., v. abs/1003.3920, 2010.

BUYYA , R.; ABRAMSON , D.; GIDDY , J.; STOCKINGER , H. Economic models for resource management and scheduling in grid computing. Wiley Press, 2002, p. 1507–1542.

BUYYA, Rajkumar; BROBERG, James; GOSCISNSKI, Andrzej M. Cloud Computing: Principles and Paradigms. John Wiley and Sons: San Francisco, 2011.

CAROLAN , J.; GAEDE , S.; BATY , J.; BRUNETTE , G.; LICHT , A.; REMMELL, J.; T UCKER , L.; WEISE , J. Sun microsystems - introduction to cloud computing architecture.
In: White paper 1st edition, pp.9-12, june 2009. Retrieved in December 07, 2009, from: http://www.sun.com/featured-articles/CloudComputing.pdf, 2009.

CASAVANT, Thomas L.; KUHL, Jon G. A taxonomy of scheduling in general-purpose distributed computing systems. IEEE Transactions on Software Engineering, New York, v. 14 n. 2, p.141-154, Feb. 1988.

CHIEU , T. C.; MOHINDRA , A.; KARVE , A. A.; SEGAL , A. Dynamic scaling of web applications in a virtualized cloud computing environment. In: ICEBE ’09: Proceedings of the 2009 IEEE International Conference on e-Business Engineering, Washington, DC, USA.

EL-REWINI, H; ABD-EL-BARR, M. Advanced Computer Architecture and Parallel Processing. [S.l.]: Wiley-Interscience, 2005. (Wiley Series on Parallel and Distributed Computing, ISBN 0-471-46740-5).

TOPCUOUGLU, H; HARIRI, S; WU, M. Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems, 13(3):260–274, 2002

IBRAHIM , S.; HE , B.; JIN , H. Towards pay-as-you-consume cloud computing. In: Services Computing (SCC), 2011 IEEE International Conference on, 2011, p. 370 –377 IEEE Computer Society, 2009, p. 281–286.

HE, Z; HE, Y. *Analysis on the security of cloud computing*. Proc. Spie, Qingdao, China, n., p.7752-775204, 2011.

HINS, M; MIERS, C; PILLON, M, A; KOSLOVSKI, G, P; Um Modelo de Custo para Nuvens IaaS baseado no Consumo de Energia de Máquinas Virtuais. XII Brazilian Symposium on Information Systems, Florianópolis, SC, May 17-20, 2016.

IBRAHIM, S.; HE, B.; JIN, H. Towards pay-as-you-consume cloud computing. In: IEEE. Services Computing (SCC), 2011 IEEE International Conference on. [S.l.], 2011. p. 370– 377.

JONES, M, T; Anatomia de um Hypervisor Linux. Uma introdução ao KVM e ao Lguest. Developer Works. 2009.

LAUREANO, M.; MAZIERO, C. Virtualização: Conceitos e aplicações em segurança. Livro-Texto de Minicursos SBSeg, p. 1–50, 2008.

MELL , P.; GRANCE , T. The NIST Definition of Cloud Computing. Relatório Técnico, 2011. Disponível em http://www.csrc.nist.gov/groups/SNS/cloud-computing/.

MENEZES, D. et al. Scheduler simulation using ispd, an iconic-based computer grid simulator. In: Computers and Communications (ISCC), 2012 IEEE Symposium on. [S.l.: s.n.], 2012. p. 000637 –000642. ISSN 1530-1346.

LEE , Y. C.; WANG , C.; ZOMAYA , A. Y.; ZHOU , B. B. scheduling in clouds. Profit-driven service request In: Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on, 2010, p. 15 –24.

LIU , Q.; WENG , C.; LI , M.; LUO , Y. An in-vm measuring framework for increasing virtual machine security in clouds. Security Privacy, IEEE, v. 8, n. 6, p. 56 –62, 2010.

JONES, M. T. Anatomia de uma nuvem de software livre. Disponível em: <http://www.ibm.com/developerworks/br/opensource/library/os-cloud-anatomy/>. Acesso em: 09 set. 2017.

KHAN, D, H. Efficient Virtual Machine Scheduling in Cloud Computing. International Journal of Computer Science and Mobile Computing, Vol.3 Issue.5, May- 2014, pg. 444-453

MAHARANA, D; SAHOO, B; SETHI, S. Energy-Efficient Real-Time Tasks Scheduling in Cloud Data Centers. National Conference on Next Generation Computing and its Applications in Science & Technology , (NGCAST)-2016, IGIT, Sarang.

MENKEN, I.; BLOKDIJK, G. Cloud Computing Foundation Complete Certification Kit Study Guide Book and Online Course. [S.l.]: Emereo Pty Ltd, 2010.

MUTHUCUMARU, M.; SHOUKAT, A.; HOWARD, J. S. DEBRA, H.; FREND, R. Dynamic matching and scheduling of a class of independent tasks onto

heterogeneous computing systems. Em Proceedings of the Eighth Heterogeneous Computing Workshop, páginas 30–44. IEEE Computer Society, 1999.

OPENSTACK. What is OpenStack? Disponível em: <https://docs.openstack.org/essex/>. Acesso em: 23 set. 2017.

PEIXOTO, M, L, M. Oferecimento de QoS para computação em nuvens por meio de metaescalonamento. Tese apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USPUniversidade de São Paulo. São Carlos 2012.

PRAJAPATI, K, D.; RAVAL, P.; KARAMTA, M.; POTDAR, M, B. Comparison of Virtual Machine Scheduling Algorithms in Cloud Computing International Journal of Computer Applications (0975 – 8887) Volume 83 – No 15, December 2013.

RAMEZANI, F.; LU, J.; HUSSAIN, F. K. Task-Based System Load Balancing in Cloud Computing Using Particle Swarm Optimization. Int J Parallel Prog (2014) 42:739–754 DOI 10.1007/s10766-013-0275-4. New York 2014.

RODRIGUEZ, M, A.; BUYYA, R. A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments. Concurrency Computat: Pract Exper. 2017;29:e4041. https://doi.org/10.1002/cpe.4041.

RIMAL , B. P.; CHOI , E.; LUMB , I. A taxonomy and survey of cloud computing systems. In: NCM ’09: Proceedings of the 2009 Fifth International Joint Conference on INC, IMS and IDC, Washington, DC, USA: IEEE Computer Society, 2009, p. 44–51.

SABAHI, F. Cloud computing security threats and responses. Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on May 2011.

SAKELLARIOU, R; ZHAO, H; TSIAKKOURI, E; DIKAIAKOS, D. Scheduling workflows with budget constraints. Em Sergei Gorlatch e Marco Danelutto, editors, Integrated Research in Grid Computing, páginas 189–202. Springer US, 2007. 16, 27, 28, 35.

SASIKALA, P. Cloud computing: present status and future implications. Disponível em:

<http://www.inderscience.com/storage/f123101246118579.pdf>. Acesso em: 10 set. 2017.

SOTIRIADIS , S.; BESSIS , N.; ANTONOPOULOS , N.survey of meta-scheduling approaches. Towards inter-cloud schedulers: A In: P2P, Parallel, Grid, Cloud and Internet.

SYSTEMS, Eucalyptus. WHAT IS EUCALYPTUS. Disponível em: <http://www.eucalyptus.com/learn/what-is-eucalyptus>. Acesso em: 26 abr. 2017. Computing (3PGCIC), 2011 International Conference on, 2011, p. 59 –66.

VCL, Apache. Apache VCL. Disponível em: <https://vcl.apache.org/>. Acesso em: 26 abr. 2017.

TANEMBAUM, A.; WOODHULL, A. “Sistemas Operacionais, Projeto e Implementação”. Tradução João Tortello. 3 ed. Porto Alegre: Bookman, 2008, 992p.

TEIXEIRA, E, C. Informações de suporte ao escalonamento de workflows científicos para a execução em plataformas de computação em nuvem. Instituto de Matemática e Estatística. Universidade de São Paulo. São Paulo SP 2016.

THE 3 rd International Caucasus Universities Association Graduate Students Symposium, Ardabil, Iran, 2016.

VIEIRA, C, C, A., Um Modelo de Escalonamento de Requisições de Máquinas Virtuais em Provedores de IaaS Considerando Diferentes Requisitos dos Usuários. Universidade Estadual de Campinas. Campinas São Paulo Brasil, 2016.

VADHIYAR , S. S.; DONGARRA , J. J. A metascheduler for the grid. In: HPDC ’02: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing, Washington, DC, USA: IEEE Computer Society, 2002, p. 343.

WANG W; NIU, D; BAOCHUN, L; LIANG, B. Dynamic cloud resource reservation Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on , pages 400#409, July 2013.

WEI, Y.; BLAKE, M. B. Proactive virtualized resource management for service workflows in the cloud. Computing, Springer, p. 1–16, 2014.

YAN, Y; CHAPMAN, B. “A Feature-Rich Workflow Description Language that Supports Resource Co-allocations”. In (L. Grandinetti, Ed.) High Performance Computing and Grids in Action by IOS Press, Amsterdam, in the Series "Advances in Parallel Computing”, 2008.

YOU , X; XU, X; WAN, J; JIANG, C. Analysis and evaluation of the scheduling algorithms in virtual environment. In: Embedded Software and Systems, 2009. ICESS ’09. International Conference on, 2009, p. 291 –296

YU, J; BUYYA, R. A budget constrained scheduling of workflow applications on utility grids using genetic algorithms. Workflows in Support of Large-Scale Science, pages 1–10, 2006. 9, 11