OpenStack Neat – A Framework for Dynamic Consolidation of Virtual Machines: A Blueprint

Anton Beloglazov

28th of July 2012

Contents

1 Summary

OpenStack Neat is a project intended to provide an extension to OpenStack implementing dynamic consolidation of Virtual Machines (VMs) using live migration. The major objective of dynamic VM consolidation is to improve the utilization of physical resources and reduce energy consumption by re-allocating VMs using live migration according to their real-time resource demand and switching idle hosts to the sleep mode. For example, assume that two VMs are placed on two different hosts, but the combined resource capacity required by the VMs to serve the current load can be provided by just one of the hosts. Then, one of the VMs can be migrated to the host serving the other VM, and the idle host can be switched to the sleep mode to save energy.

Apart from consolidating VMs, the system should be able to react to increases in the resource demand and deconsolidate VMs when necessary to avoid performance degradation. In general, the problem of dynamic VM consolidation can be split into 4 sub-problems:

1. Deciding when a host is considered to be underloaded, so that all the VMs should be migrated out, and the host should be switched to a low-power mode, such as the sleep mode.
2. Deciding when a host is considered to be overloaded, so that some VMs should be migrated from the host to other hosts to avoid performance degradation.
3. Selecting VMs, which should be migrated from an overloaded host out of the full set of the VMs currently served by the host.

4. Placing VMs selected for migration to other active or re-activated hosts.

This work is a part of PhD research conducted within the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne. The problem of dynamic VM consolidation considering Quality of Service (QoS) constraints has been studied from the theoretical perspective and algorithms addressing the sub-problems listed above have been proposed [1], [2]. The algorithms have been evaluated using CloudSim and real-world workload traces collected from more than a thousand PlanetLab VMs hosted on servers located in more than 500 places around the world.

The aim of the OpenStack Neat project is to provide an extensible framework for dynamic consolidation of VMs within OpenStack environments. The framework should provide an infrastructure enabling the interaction of components implementing the 4 decision-making algorithms listed above. The framework should allow configuration-driven switching of implementations of the decision-making algorithms. The implementation of the framework will include the algorithms proposed in our previous works [1], [2].

2 Release Note

The functionality covered by this project will be implemented in the form of services separate from the core OpenStack services. The services of this project will interact with the core OpenStack services using their public APIs. It will be required to create a new Keystone user within the `service` tenant. The project will also require a new MySQL database for storing information about the host configuration, VM placement, and CPU utilization by the VMs. The project will provide a script for automated initialization of the database. The services provided by the project will need to be run on the management as well as compute hosts.

3 Rationale

The problem of data centers is high energy consumption, which has risen by 56% from 2005 to 2010, and in 2010 accounted to be between 1.1% and 1.5% of the global electricity use [3]. Apart from high operating costs, this results in substantial carbon dioxide (CO_2) emissions, which are estimated to be 2% of the global emissions [4]. The problem has been partially addressed by improvements in the physical infrastructure of modern data centers. As reported by the Open Compute Project, Facebook's Oregon data center achieves a Power Usage Effectiveness (PUE) of 1.08, which means that approximately 93 of the data center's energy consumption are consumed by the computing resources. Therefore, now

it is important to focus on the resource management aspect, i.e. ensuring that the computing resources are efficiently utilized to serve applications.

Dynamic consolidation of VMs has been shown to be efficient in improving the utilization of data center resources and reducing energy consumption, as demonstrated by numerous studies [5–16]. In this project, we aim to implement an extensible framework for dynamic VM consolidation specifically targeted at the OpenStack platform.

4 User stories

- As a Cloud Administrator or Systems Integrator, I want to support dynamic VM consolidation to improve the utilization of the data center's resources and reduce the energy consumption.
- As a Cloud Administrator, I want to provide QoS guarantees to the consumers, while applying dynamic VM consolidation.
- As a Cloud Administrator, I want to minimize the price of the service provided to the consumers by reducing the operating costs through the reduced energy consumption.
- As a Cloud Administrator, I want to decrease the carbon dioxide emissions into the environment by reducing the energy consumption by the data center's resources.
- As a Cloud Service Consumer, I want to pay the minimum price for the service provided through the minimized energy consumption of the computing resources.
- As a Cloud Service Consumer, I want to use Green Cloud resources, whose provider strives to reduce the impact on the environment in terms of carbon dioxide emissions.

5 Assumptions

Nova uses a *shared storage* for storing VM instance data, thus supporting *live migration* of VMs.

6 Design

The system is composed of a number of components and data stores, some of which are deployed on the compute hosts, and some on the management host (Figure 1). In the following sections, we discuss the design and interaction of the components, as well as the specification of the data stores.

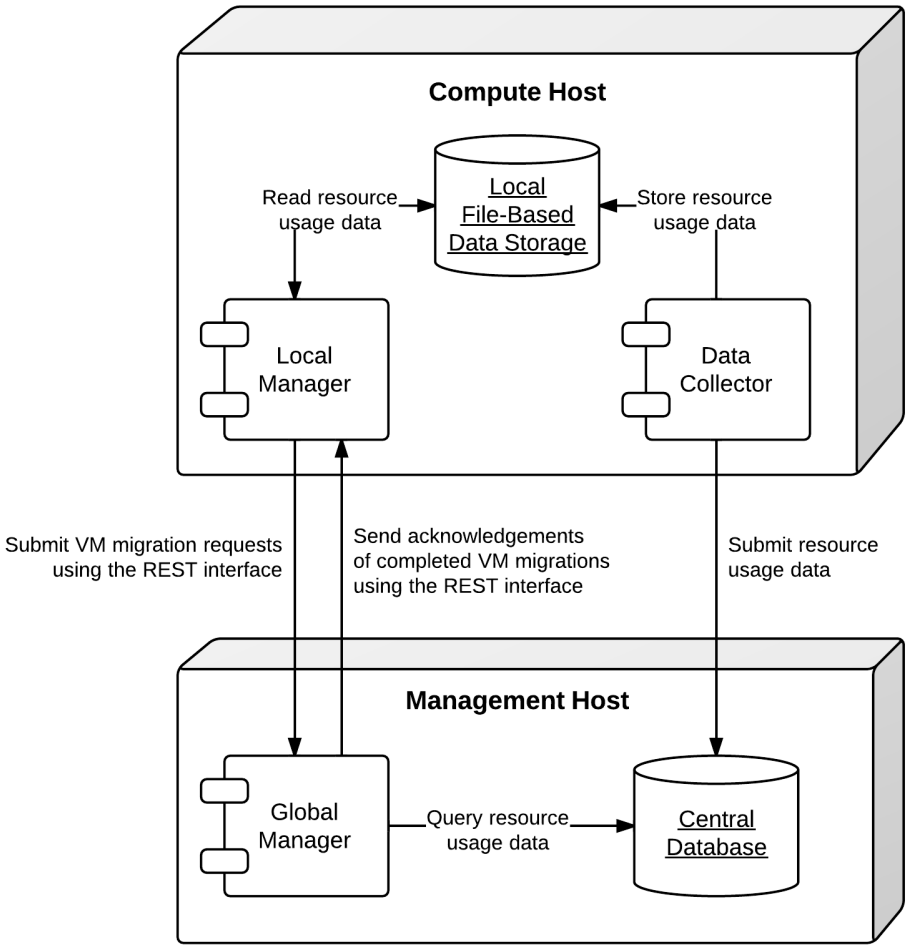

Figure 1: The deployment diagram

6.1 Components

As shown in Figure 1, the system is composed of three main components:

- *Global manager* – a component that is deployed on the management host and makes global management decisions, such as mapping VM instances on hosts, and initiating VM migrations.
- *Local manager* – a component that is deployed on every compute host and makes local decisions, such as deciding that the host is underloaded or overloaded, and selecting VMs to migrate to other hosts.
- *Data collector* – a component that is deployed on every compute host and is responsible for collecting data about resource usage by VM instances, as well as storing these data locally and submitting the data to the central database.

6.1.1 Global Manager

The global manager is deployed on the management host and is responsible for making VM placement decisions and initiating VM migrations. It exposes a REST web service, which accepts requests from local managers. The global manager processes only one type of requests – reallocation of a set of VM instances. As shown in Figure 2, once a request is received, the global manager invokes a VM placement algorithm to determine destination hosts to migrate the VMs to. Once a VM placement is determined, the global manager submits a request to the Nova API to migrate the VMs. When the required VM migrations are completed, the global manager sends an acknowledgment request to the local manager that has originated the VM migration to update its local VM metadata. The global manager is also responsible for switching idle hosts to the sleep mode, as well as re-activating hosts when necessary.

VM Placement. The global manager is agnostic of a particular implementation of the VM placement algorithm in use. The VM placement algorithm to use can be specified in the configuration file described later. A VM placement algorithm can the Nova API to obtain the information about host characteristics and current VM placement. If necessary, it can also query the central database to obtain the historical information about the resource usage by the VMs.

REST API. The global manager exposes a REST web service (REST API) for accepting VM migration requests from local managers. The service URL is defined according to configuration options defined in `/etc/neat/neat.conf`, which is discussed further in the paper. The two relevant options are:

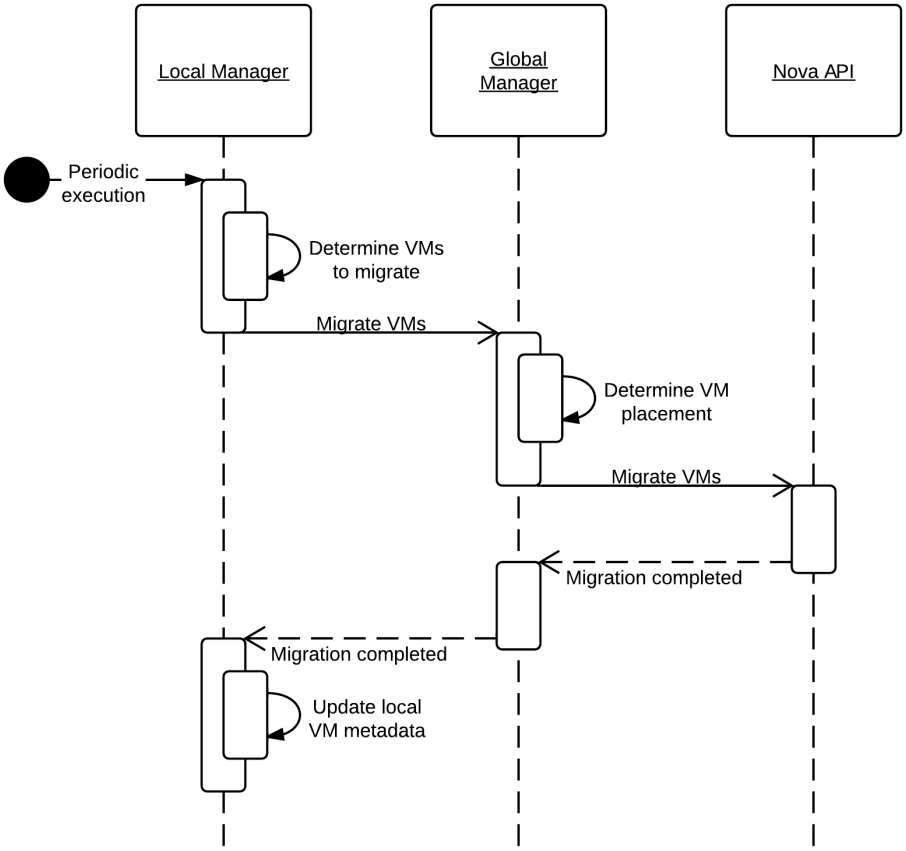

Figure 2: The global manager: a sequence diagram

- `global_manager_host` – the name of the host running the global manager;
- `global_manager_port` – the port of the REST web service exposed by the global manager.

The service URL is composed as follows:

```
http://<global_manager_host>:<global_manager_port>/
```

Since the global manager processes only a single type of requests, it exposes only one resource: `/`. The resource is accessed using the method `PUT`, which initiates the VM reallocation process. This service requires the following parameters:

- `admin_tenant_name` – the admin tenant name of Neat's admin user registered in Keystone. This parameter is not used to authenticate in any OpenStack service, rather it is used to authenticate the client making a request as being allowed to access the web service.
- `admin_user` – the admin user name of Neat's admin user registered in Keystone. This parameter is not used to authenticate in any OpenStack service, rather it is used to authenticate the client making a request as being allowed to access the web service.
- `admin_password` – the admin password of Neat's admin user registered in Keystone. This parameter is not used to authenticate in any OpenStack service, rather it is used to authenticate the client making a request as being allowed to access the web service.
- `vm_uuids` – a coma-separated list of UUIDs of the VMs required to be migrated.

If the provided credentials are correct and the `vm_uuids` parameter includes a list of UUIDs of existing VMs in the correct format, the service responses with the HTTP status code `200 OK`.

The service uses standard HTTP error codes to response in cases of errors detected. The following error codes are used:

- `400` – bad input parameter: incorrect or missing parameters;
- `401` – unauthorized: user credentials are missing;
- `403` – forbidden: user credentials do not much the ones specified in the configuration file;
- `405` – method not allowed: the request is made with a method other than the only supported `PUT`;

- `422` – unprocessable entity: one or more VMs could not be found using the list of UUIDs specified in the `vm_uuids` parameter.

Once the requested VM migrations are completed, the global manager sends an acknowledgment request to the local manager that has originated the VM migration using its REST API described later.

6.1.2 Local Manager

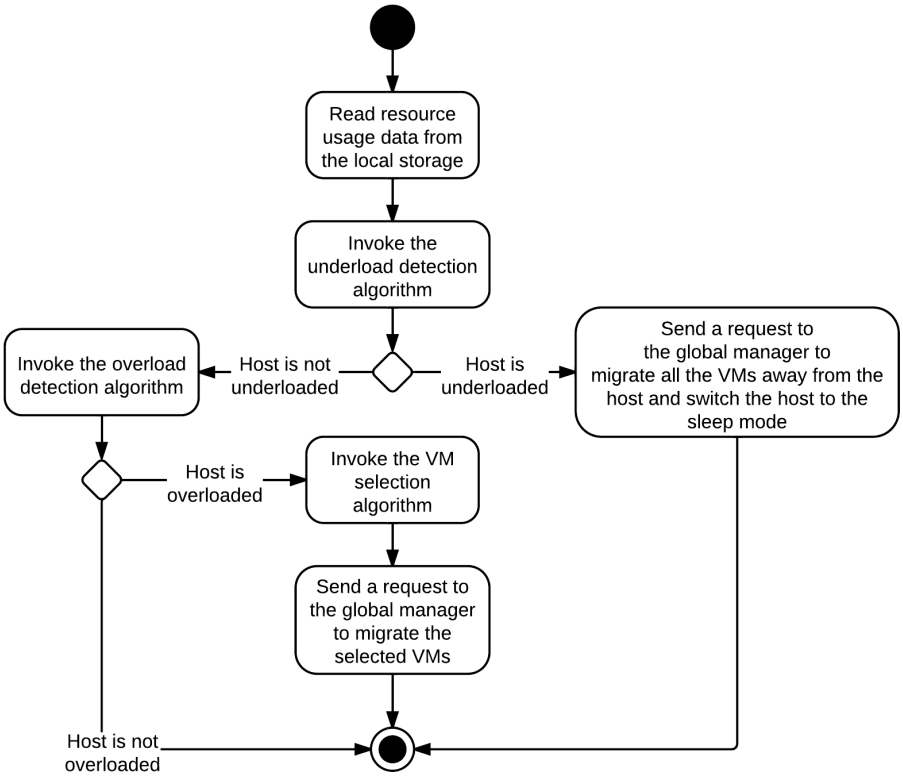

Figure 3: The local manager: an activity diagram

The local manager component is deployed on every compute host and is invoked periodically to determine when it necessary to reallocate VM instances from the host. A high-level view of the workflow performed by the local manager is shown in Figure 3. First of all, it reads from the local storage the historical data about the resource usage by the VMs stored by the data collector described in the next section. Then, the local manager invokes the specified in the configuration underload detection algorithm to determine whether the host is underloaded. If the host is underloaded, the local manager sends a request to the global

manager's REST interface to migrate all the VMs from the host and switch the host to the sleep mode.

If the host is not underloaded, the local manager proceeds to invoking the specified in the configuration overload detection algorithm. If the host is overloaded, the local manager invokes the configured VM selection algorithm to select the VMs to migrate from the host. Once the VMs to migrate from the host are selected, the local manager sends a request to the global manager's REST interface to migrate the selected VMs from the host.

The local manager also exposes a REST web service to receive acknowledgments from the global manager when the requested VM migrations are completed. Upon receiving an acknowledgment, the local manager removes from the local data store the data about the resource usage of the VMs migrated from the host.

Similarly to the global manager, the local manager can be configured to use specific underload detection, overload detection, and VM selection algorithm using the configuration file discussed further in the paper.

Underload Detection. Underload detection is done by a specified in the configuration underload detection algorithm. The algorithm has a pre-defined interface, which allows substituting different implementations of the algorithm. The configured algorithm is invoked by the local manager and accepts the historical data about the resource usage by the VMs running on the host as an input. An underload detection algorithm returns a decision of whether the host is underloaded.

Overload Detection. Overload detection is done by a specified in the configuration overload detection algorithm. Similarly to underload detection, all overload detection algorithms implement a pre-defined interface to enable configuration-driven substitution of difference implementations. The configured algorithm is invoked by the local manager and accepts the historical data about the resource usage by the VMs running on the host as an input. An overload detection algorithm returns a decision of whether the host is overloaded.

VM Selection. If a host is overloaded, it is necessary to select VMs to migrate from the host to avoid performance degradation. This is done by a specified in the configuration VM selection algorithm. Similarly to underload and overload detection algorithms, different VM selection algorithm can plugged in according to configuration. A VM selection algorithm accepts the historical data about the resource usage the VMs running on the host and returns a set of VMs to migrate from the host.

REST API. The local manager exposes a REST web service (REST API) for accepting requests from the global manager that acknowledge the completion of a

requested VM migration. The service URL is defined according to a configuration option defined in `/etc/neat/neat.conf`. The relevant option is:

- `local_manager_port` – the port of the REST web service exposed by the local manager.

The service URL is composed as follows:

```
http://<ip>:<global_manager_port>/
```

Where `<ip>` is replaced by the IP address of the host, where the local manager is running.

Since the local manager processes only a single type of requests, it exposes only one resource: `/`. The resource is accessed using the method `PUT`, which notifies the local manager that a number of VMs have been migrated from the host and the metadata needs to be updated. This service requires the following parameters:

- `admin_tenant_name` – the admin tenant name of Neat's admin user registered in Keystone. This parameter is not used to authenticate in any OpenStack service, rather it is used to authenticate the client making a request as being allowed to access the web service.
- `admin_user` – the admin user name of Neat's admin user registered in Keystone. This parameter is not used to authenticate in any OpenStack service, rather it is used to authenticate the client making a request as being allowed to access the web service.
- `admin_password` – the admin password of Neat's admin user registered in Keystone. This parameter is not used to authenticate in any OpenStack service, rather it is used to authenticate the client making a request as being allowed to access the web service.
- `vm_uuids` – a coma-separated list of UUIDs of the VMs required to be migrated.

If the provided credentials are correct and the `vm_uuids` parameter includes a list of UUIDs of existing VMs in the correct format, the service responses with the HTTP status code `200 OK`.

The service uses standard HTTP error codes to response in cases of errors detected. The following error codes are used:

- `400` – bad input parameter: incorrect or missing parameters;

- `401` – unauthorized: user credentials are missing;
- `403` – forbidden: user credentials do not much the ones specified in the configuration file;
- `405` – method not allowed: the request is made with a method other than the only supported `PUT`;
- `422` – unprocessable entity: one or more VMs could not be found using the list of UUIDs specified in the `vm_uuids` parameter.

Once a request is received, the local manager deletes the files containing the historical data on the resource usage by the VMs specified by the provided list of UUIDs.

6.1.3 Data Collector

The data collector is deployed on every compute host and is executed periodically to collect the CPU utilization data for each VM running on the host and stores it in the local file-based data store. The data is collected in average number of MHz consumed by a VM during the last measurement interval. The CPU usage data are stored as integers. This data format is portable: the collected values can be converted to the CPU utilization for any host or VM type, supporting heterogeneous hosts and VMs.

The actual data is obtained from Libvirt in the form of the CPU time consumed by a VM to date. Using the CPU time collected at the previous time frame, the CPU time for the past time interval is calculated. According to the CPU frequency of the host and the length of the time interval, the CPU time is converted into the required average MHz consumed by the VM over the last time interval. The collected data are stored both locally and submitted to the central database.

6.2 Data Stores

As shown in Figure 1, the system contains two types of data stores:

- *Central database* – a database deployed on the management host.
- *Local file-based data storage* – a data store deployed on every compute host and used for storing resource usage data to use by local managers.

The details about the data stores are given in the following subsections.

6.2.1 Central Database

The central database is used for storing historical data about the resource usage by the VMs running on all the compute hosts. The database is populated by data collectors deployed on the compute hosts. The data is consumed by VM placement algorithms. The database contains two tables: `vms` and `vm_resource_usage`.

The `vms` table is used for storing the mapping between UUIDs of VMs and the internal database IDs:

```
CREATE TABLE vms (
    # the internal ID of a VM
    id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
    # the UUID of the VM
    uuid CHAR(36) NOT NULL,
    PRIMARY KEY (id)
) ENGINE=MyISAM;
```

The `vm_resource_usage` table is used for storing the data about the resource usage by VMs:

```
CREATE TABLE vm_resource_usage (
    # the ID of the record
    id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
    # the id of the corresponding VM
    vm_id BIGINT UNSIGNED NOT NULL,
    # the time of the data collection
    timestamp TIMESTAMP NOT NULL,
    # the average CPU usage in MHz
    cpu_mhz MEDIUMINT UNSIGNED NOT NULL,
    PRIMARY KEY (id)
) ENGINE=MyISAM;
```

6.2.2 Local File-Based Data Store

The data collector stores the resource usage information locally in files in the `<local_data_directory>/vm`, where `<local_data_directory>` is defined in the configuration file discussed further in the paper. The data for each VM is stored in a separate file named according to the UUID of the corresponding VM. The format of files is a new line separated list of integers representing the CPU consumption by the VMs in MHz.

6.3 Configuration File

The configuration of OpenStack Neat is stored in `/etc/neat/neat.conf` in the standard INI format using the `#` character for denoting comments. The configuration includes the following options:

- `sql_connection` – the host name and credentials for connecting to the MySQL database specified in the format supported by SQLAlchemy;
- `admin_tenant_name` – the admin tenant name for authentication with Nova using Keystone;
- `admin_user` – the admin user name for authentication with Nova using Keystone;
- `admin_password` – the admin password for authentication with Nova using Keystone;
- `global_manager_host` – the name of the host running the global manager;
- `global_manager_port` – the port of the REST web service exposed by the global manager;
- `local_manager_port` – the port of the REST web service exposed by the local manager;
- `local_data_directory` – the directory used by the data collector to store the data on the resource usage by the VMs running on the host, the default value is `/var/lib/neat`.

6.4 TODO

- Define REST APIs for the Global and Local Managers
- Find out how to remotely switch hosts on or off

7 Implementation

This section should describe a plan of action (the "how") to implement the changes discussed. Could include subsections like:

7.1 Libraries

- pyqcy – a QuickCheck-like testing framework for Python.
- PyContracts – a Python library for Design by Contract (DbC).
- SQLAlchemy – a Python SQL toolkit and Object Relational Mapper, it is used by OpenStack.
- Bottle – a micro web-framework for Python, authentication using the same credentials using for Nova.
- python-novaclient – a python client API to Nova.
- Sphinx – a documentation generator for Python.

7.2 UI Changes

Should cover changes required to the UI, or specific UI that is required to implement this

7.3 Code Changes

Code changes should include an overview of what needs to change, and in some cases even the specific details.

7.4 Migration

Include:

- data migration, if any
- redirects from old URLs to new ones, if any
- how users will be pointed to the new way of doing things, if necessary.

8 Test/Demo Plan

This need not be added or completed until the specification is nearing beta.

9 Unresolved issues

This should highlight any issues that should be addressed in further specifications, and not problems with the specification itself; since any specification with problems cannot be approved.

10 BoF agenda and discussion

Use this section to take notes during the BoF; if you keep it in the approved spec, use it for summarising what was discussed and note any options that were rejected.

11 References

[1] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers," *Concurrency and Computation: Practice and Experience (CCPE)*, 2012.

[2] A. Beloglazov and R. Buyya, "Managing Overloaded Hosts for Dynamic Consolidation of Virtual Machines in Cloud Data Centers Under Quality of Service Constraints," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2012 (under review).

[3] J. Koomey, *Growth in data center electricity use 2005 to 2010*. Oakland, CA: Analytics Press, 2011.

[4] Gartner Inc., *Gartner estimates ICT industry accounts for 2 percent of global CO2 emissions*. Gartner Press Release (April 2007).

[5] R. Nathuji and K. Schwan, "VirtualPower: Coordinated power management in virtualized enterprise systems," *ACM SIGOPS Operating Systems Review*, vol. 41, pp. 265–278, 2007.

[6] A. Verma, P. Ahuja, and A. Neogi, "pMapper: Power and migration cost aware application placement in virtualized systems," in *Proc. of the 9th ACM/IFIP/USENIX Intl. Conf. on Middleware*, 2008, pp. 243–264.

[7] X. Zhu, D. Young, B. J. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, and others, "1000 Islands: Integrated capacity and workload management for the next generation data center," in *Proc. of the 5th Intl. Conf. on Autonomic Computing (ICAC)*, 2008, pp. 172–181.

[8] D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Turicchi, and A. Kemper, "An integrated approach to resource pool management: Policies, efficiency and

quality metrics," in *Proc. of the 38th IEEE Intl. Conf. on Dependable Systems and Networks (DSN)*, 2008, pp. 326–335.

[9] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Resource pool management: Reactive versus proactive or lets be friends," *Computer Networks*, vol. 53, pp. 2905–2922, 2009.

[10] VMware Inc., "VMware Distributed Power Management Concepts and Use," *Information Guide*, 2010.

[11] G. Jung, M. A. Hiltunen, K. R. Joshi, R. D. Schlichting, and C. Pu, "Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures," in *Proc. of the 30th Intl. Conf. on Distributed Computing Systems (ICDCS)*, 2010, pp. 62–73.

[12] W. Zheng, R. Bianchini, G. J. Janakiraman, J. R. Santos, and Y. Turner, "JustRunIt: Experiment-based management of virtualized data centers," in *Proc. of the 2009 USENIX Annual Technical Conf.*, 2009, pp. 18–33.

[13] S. Kumar, V. Talwar, V. Kumar, P. Ranganathan, and K. Schwan, "vManage: Loosely coupled platform and virtualization management in data centers," in *Proc. of the 6th Intl. Conf. on Autonomic Computing (ICAC)*, 2009, pp. 127–136.

[14] B. Guenter, N. Jain, and C. Williams, "Managing Cost, Performance, and Reliability Tradeoffs for Energy-Aware Server Provisioning," in *Proc. of the 30st Annual IEEE Intl. Conf. on Computer Communications (INFOCOM)*, 2011, pp. 1332–1340.

[15] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing SLA violations," in *Proc. of the 10th IFIP/IEEE Intl. Symp. on Integrated Network Management (IM)*, 2007, pp. 119–128.

[16] A. Beloglazov, R. Buyya, Y. C. Lee, and A. Zomaya, "A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems," *Advances in Computers, M. Zelkowitz (ed.)*, vol. 82, pp. 47–111, 2011.