Resource Assignment for Large-Scale Computing Utilities using Mathematical Programming

Xiaoyun Zhu, Cipriano Santos, Julie Ward, Dirk Beyer,
Sharad Singhal
Internet Systems and Storage Laboratory
HP Laboratories Palo Alto
HPL-2003-243(R.1)
February 11, 2004*

E-mail: {xiaoyun.zhu, cipriano.santos, jward, dirk.beyer, sharad.singhal}@hp.com

utility computing, resource assignment, storage area networks, mixed integer programming

In this paper, we describe a resource assignment problem (RAP) for a large-scale computing utility, such as an Internet data center. The problem is defined as follows: For a given topology of a network consisting of switches and servers with varying capabilities, and for a given application with a distributed architecture, decide which server from the physical network should be assigned to each application component, such that the traffic-weighted average inter-server distance is minimized, and the application's processing, communication and storage requirements are satisfied without exceeding network capacity limits. This problem is first formulated as a nonlinear combinatorial optimization problem. We then describe three mixed integer programming formulations, RAP-LINI, RAP-LINII, and RAP-MCFM, as the result of different linearization techniques. These models were numerically tested using CPLEX on a number of examples, ranging from a 125-server utility data center to a set of hypothetical data centers with increasing size. In all cases and for all three models, the CPELX solver was able to find an optimal solution within reasonable amount of time. RAP-LINII is the most efficient and required the minimum solution time. RAP-MCFM has the highest complexity but is the most general in terms of its applicability to any network topology.

* Internal Accession Date Only
Approved for External Publication

Resource Assignment for Large-Scale Computing Utilities using Mathematical Programming

Xiaoyun Zhu, Cipriano Santos, Julie Ward, Dirk Beyer, Sharad Singhal

Hewlett Packard Laboratories

Palo Alto, CA 94304

{xiaoyun.zhu, cipriano.santos, jward, dirk.beyer, sharad.singhal}@hp.com

Abstract

In this paper, we describe a resource assignment problem (RAP) for a large-scale computing utility, such as an Internet data center. The problem is defined as follows: For a given topology of a network consisting of switches and servers with varying capabilities, and for a given application with a distributed architecture, decide which server from the physical network should be assigned to each application component, such that the traffic-weighted average inter-server distance is minimized, and the application's processing, communication and storage requirements are satisfied without exceeding network capacity limits. This problem is first formulated as a nonlinear combinatorial optimization problem. We then describe three mixed integer programming formulations, RAP-LINI, RAP-LINII, and RAP-MCFM, as the result of different linearization techniques. These models were numerically tested using CPLEX on a number of examples, ranging from a 125-server utility data center to a set of hypothetical data centers with increasing size. In all cases and for all three models, the CPELX solver was able to find an optimal solution within reasonable amount of time. RAP-LINII is the most efficient and required the minimum solution time. RAP-MCFM has the highest complexity but is the most general in terms of its applicability to any network topology.

Keywords: utility computing, resource assignment, storage area networks, mixed integer programming

1. Introduction

1.1 Motivation and background

Although utility computing is viewed by many as the model of computing for the future, the vision has been around for decades. The MULTICS project in the 1960s [6] had the goal of developing "a new computer system specifically organized as a prototype of a computer utility," with one of its requirements being "continuous operation analogous to that of the electric power and telephone companies." In a computing utility, computing resources and capabilities are provided to people and businesses as a service. One example of a computing utility that exists today is the Grid [9], which offers spare compute cycles to scientific and engineering applications. Another example is data center, where a large pool of IT resources are centrally managed to meet the needs of business critical enterprise applications such as enterprise resource planning applications, database applications, customer relationship management applications, and general e-commerce applications. There has been a wave of industrial initiatives to provide infrastructure and management support for such utilities, including On Demand Computing [11], HP's Utility Data Center (UDC) [22], Sun's N1 initiative [21], Microsoft's Dynamic Systems Initiative [14], and many others.

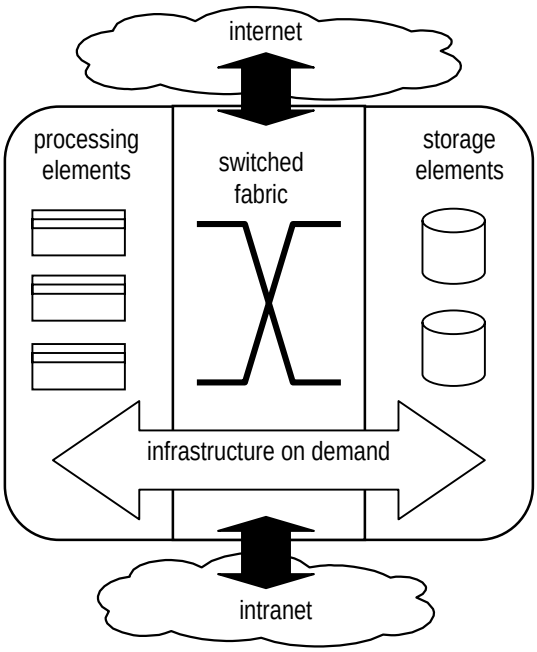

Figure 1. Architecture of a computing utility

Figure 1 is an architecture diagram for a computing utility. Such an environment can contain thousands of

servers and storage devices connected through a shared high speed network fabric. The goal is to offer "infrastructure on demand," which means compute, networking, and storage resources are provided to applications as they need them. Most of the resources will be virtualized and shared across multiple applications to achieve economies of scale and increase return on investment. The complexity of managing such an infrastructure and applications simultaneously is enormous. Automation is needed to lower operation cost and reduce human error. Well-informed capacity planning and resource provisioning are required to increase asset utilization and meet service level objectives.

When an application is deployed in a computing utility, it is allocated a partition of resources in a virtual application environment [16] to meet the specific needs of the application. As each application's real time workload varies over time, resources can be dynamically re-allocated and re-distributed among all running applications to achieve high resource utilization. In most cases, the physical identities of the allocated resources are transparent to the application due to virtualization of resources. And it is the utility provider's job to choose the right set of physical resources for each application and its components to satisfy the application's configuration and performance requirements, to avoid resource bottlenecks in the infrastructure, to achieve certain goals or enforce certain policies. We refer to this decision making process as *resource assignment*. Techniques for dealing with this process are an integral part of a resource access management framework [20] that controls the complete lifecycle of applications' access to resources in a computing utility.

In today's data centers, resource assignment is typically done by human operators, which is slow, expensive, and error prone. As the size of future computing utilities grows into the magnitude of tens of thousands of resources, the number of possibilities to provision a given application goes far beyond the tracking ability of any human. This calls for a more systematic approach for resource assignment so that it can be automated to significantly shorten application deployment cycles and minimize operator overhead. In general, a naïve scheme such as random selection or first-come-first-served may not work because there are too many consequences to any particular solution that is chosen. For instance, the compute requirements of the application may not be met by some of the servers, the latency of the application can be poor, or the cost involved may be too high, etc. In particular, since networking resources are shared among different applications and their components, it is highly likely for a network link to become a bottleneck thus degrading the performance of the applications that share this link. This, of course, has the assumption that network resources are not over-provisioned, and relatively high utilization on these resources is desired. We believe this is reasonable to assume given the current economic pressure to reduce IT cost and to increase return on investment. Therefore, resource assignment is a highly complex problem that requires more intelligent solution techniques.

Every application to be deployed in a computing utility has high-level metrics such as number of concurrent users, number of transactions per second and infrastructure cost. Usually the mapping between these requirements and the specific identities of the resources that are used to host the application is not straightforward. We believe that a two-step process is the most efficient to perform this mapping, which is shown in Figure 2. A step that we refer to as "Grounding" translates the application's high-level requirements into a "grounded application model" that represents the low-level processing, communication and storage requirements on the physical resources. This step requires a great deal of domain knowledge and experience with the specific application, and typically involves benchmarking exercises. "Resource assignment" chooses the specific instances of resources from the infrastructure. This step requires knowledge and data on the physical resources. This paper deals with the second step. Although how to do grounding effectively is an interesting research question in itself, it is not addressed here.

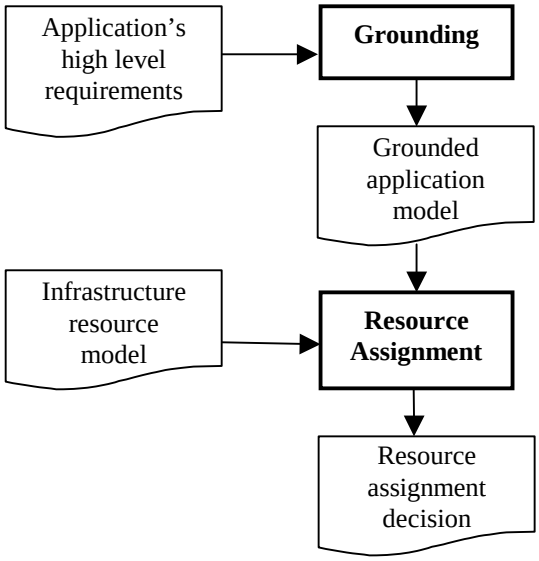

Figure 2. Application requirement mapping process

1.2 Problem statement

In this paper, we specifically study the following *resource assignment problem* (RAP): *For a given topology of a network consisting of switches and servers with varying capabilities, and for a given application with a distributed architecture and a set of requirements for processing, communication and storage; decide which*

server from the physical network should be assigned to each application component, such that the traffic-weighted average inter-server distance is minimized, and the application requirements are satisfied without exceeding network capacity limits.

This problem is first formulated as a nonlinear combinatorial problem. Shahoumian proved that it is NP-hard [19] by showing that the problem of Minimum Cut into Bounded Sets, a well-known NP-complete problem [8], can be reduced to RAP in polynomial time. There are potentially many algorithms for tackling this kind of problem, including simple heuristics such as greedy algorithms, and more evolved meta-heuristics such as Tabu search, genetic algorithms, and simulated annealing [13]. We chose *mathematical programming* (MP) [23] for the following reasons. First, MP is a common and flexible framework for modeling a large class of optimization problems. Second, there are commercially available MP solvers that have been tested extensively by both academia and industry, such as *CPLEX* [10]. This gives us the ability to separate models from solution techniques. We can focus on developing representative models for real systems, improve efficiency of problem formulations, and revise the models as assumptions change. Third, MP is a powerful tool for handling a large number of hard capacity constraints, a characteristic of the RAP problem, which is difficult to deal with using meta-heuristic type of methods. Finally, the algorithms used by *CPLEX* for solving MP problems, such as the branch and bound algorithm for integer programming (IP), have provable global optimality. And for any given period of computation time, bounds on the optimal objective function value can be provided as an indicator of the solution quality. On the other hand, one major drawback of the MP approach is that it requires someone who is well-versed in the MP language to develop a good MP model for a given problem. The solution time often depends on the efficiency of the particular formulation.

The main contribution of this paper is the formalization of resource assignment in a computing utility as a mathematical optimization problem. In addition, reformulating the problem as various *mixed integer programming* (MIP) problems enables us to use an off-the-shelf solver such as *CPLEX* for finding optimal solutions. Finally, the multicommodity flow formulation of RAP makes it applicable to resource assignment in networks with arbitrary topology.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 describes the mathematical model for the application. Section 4 describes the resource model for a computing utility. Section 5 presents the complete optimization problem for RAP. In Section 6, we present three MIP formulations for the original nonlinear combinatorial problem. Section 7 describes numerical experiments on a set of examples. The results are presented to demonstrate the efficiency of the solution techniques and the complexity of the three models. Section 8 offers conclusion and future directions.

2. Related Work

There is a rich literature on resource assignment problems in a wide range of computer systems or networking systems. The terminology comes in many forms, such as “device selection,” “application placement,” “node placement”, etc., which all refer to the process of choosing the right physical resources for hosting certain computing tasks. Each piece of work may differ from ours in certain aspects. For example, it may be assigning different kinds of resources in a different environment, such as nodes in a supercomputer cluster or spare workstations in a grid. In addition, it may focus on applications/jobs of different nature with different characteristics in their resource requirements, such as short-lived batch processing jobs. The optimization criterion could range from application performance, economic concerns, to certain utility functions. Finally, the solution techniques can vary significantly.

A large amount of prior work in this area dealt with uncapacitated networks. This applies to the well-known facility location problem [3], a widely studied quadratic assignment problem, where n facilities are assigned onto n sites such that the average transportation cost between sites is minimized. Another example is the conventional node placement optimization (NPO) [4], where the job is to allocate traffic matrix source/destinations to nodes in a multi-processor interconnection architecture such that the mean traffic-weighted inter-nodal distance is minimized. Both problems are difficult to solve optimally, and many heuristic approaches have been exploited to find near-optimal solutions. Our RAP problem involves a similar decision problem and has essentially the same objective function, but is even harder in that any assignment solution has to satisfy a set of capacity constraints.

The resource allocation problem studied in [12] does consider a capacitated network. The problem is: given a set of jobs that require computing and bandwidth resources and generate profits, select the feasible subset of jobs that maximizes profits and decide placement of these jobs onto a network of compute nodes. This paper contributes to the current grid-computing or peer-to-peer systems research in providing “bandwidth guarantee” in resource allocation or job scheduling. However, it is more suitable for large scientific and engineering jobs that can rely on aggregated resources from multiple nodes to fulfill a single task. It is not a model with sufficient information for more complex enterprise applications.

The resource assignment problem was originally defined in [24], where the task was to assign physical

servers in a tree network to logical servers in a multi-tier application architecture. In [17,18] the same problem was reformulated as a series of MIP and quadratic programming (QP) problems that were solved using commercial solvers. The solution technique proved more scalable than the enumerative approach used in [24]. The RAP problem defined in this paper differs from the above in the following aspects. First, we now deal with more general distributed applications whose component architecture can be an arbitrary graph. Second, the utility infrastructure we study here includes a storage area network (SAN), which has become increasingly important in modern data centers, but was not present in the earlier work. This addition allows our work to be applied to applications that access data in centralized storage devices through a SAN. Third, the tree network in the earlier problem has a hierarchical structure, where servers only appear at the bottom layer. This restrictive assumption has been removed. Furthermore, the multicommodity flow model in this paper allows the LAN topology to be an arbitrary graph as well.

A decision related to server assignment is application data placement in storage devices, commonly known as the *file assignment problem* (FAP). This problem has been studied in depth using optimization approaches. Dowdy and Foster gave a unified view of various models people had developed for FAP and provided detailed comparison of their respective contributions and suitable solution techniques [7]. However, none of these models had the concept of separated compute nodes and storage nodes and the notion of a storage area network with capacity limits. One piece of more recent work in this area is the *Ergastulum* project [2] at HP Labs. For a given storage access workload, Ergastulum solves the data placement problem as well as the storage system configuration problem at the same time. Compared to the storage related models in our RAP formulation, Ergastulum uses a more sophisticated storage workload description and contains a more detailed view of each storage device. However, Ergastulum does not take into account capacity constraints in the SAN when it heuristically searches for optimal file mapping onto disk arrays. Therefore, Ergastulum does a better job optimizing the performance of storage devices but ignores the storage network performance. The latter, we believe, is a practical concern for many storage systems. A full blown optimization problem encompassing file placement and server assignment seems appealing, but is likely to be computationally intractable. As a first step, we choose to separate these two problems. We can use Ergastulum or any other FAP solver to generate a near-optimal file placement decision for a given application workload. It can then be provided as an input to our resource assignment problem, where we focus on optimal server selection while meeting SAN capacity constraints.

3. The Application Model

This section describes the mathematical model for the application architecture obtained in the grounding process, which serves as an input to the resource assignment problem.

3.1 A component-graph based model

An application can be characterized by a set of components that communicate with one another in a certain way. It can be represented by a directed graph G(C, L), where each node $c \in C$ represents an application component, and each directed edge $l = (c, c') \in L$ is an ordered pair of component nodes, representing communication from component c to component c'. The matrix T is defined to characterize the traffic pattern of the application. Each element $T_{cc'}$ represents the maximum amount of traffic going from component c to component c'. $T_{cc'} = 0$ if an edge (c,c') does not exist, indicating no traffic flows from component c to component c'. The component-graph based application architecture is illustrated in Figure 3.

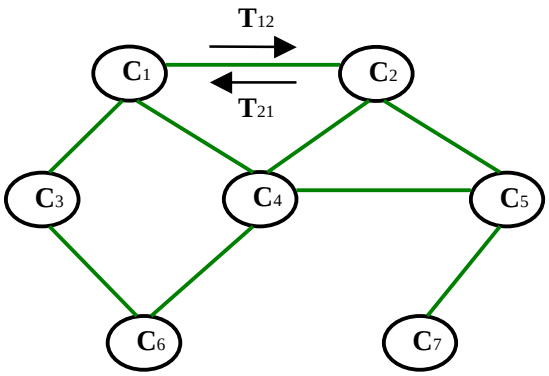

Figure 3. Component-graph based application model

Each application component has requirements on the type of servers on which it can be hosted. Let P to be the set of server attributes (or properties) that are of interest to a particular application, such as processor type, processor speed, number of processors, memory size, disk space, and so on. Then for each attribute $p \in P$ and each application component $c \in C$, the requirement is characterized by a set $VREQ_{cp}$, which contains the permissible values of attribute p for component c. This set may be either discrete or continuous. For example, an application component may require a server's processor type to be in {SPARC, PA_RISC}, and its processor speed to be in an interval [500, 1000] (in MHz).

The multi-tier application architecture studied in [17,18,24] can be considered a special case of the above

component-based model, where a tier consists of multiple application components of similar functionality. For a given network of compute nodes, instead of assigning the right server to each application component, the task there is to assign the right set of servers to each tier, which results in a smaller problem with fewer variables. When the application to be deployed does have a tiered structure, we should consider using the multi-tier model so that the solution time can be shortened.

3.2 Model for storage requirements

Assume that data for an application can be divided into a set of "files". Here we use the abstract notion of a file to represent a logically contiguous chunk of data that may be accessed by application components. The storage access pattern of all the components can be represented by a bipartite graph as shown in

Figure **4**. The example illustrates that the mapping between an application component and a file is not one-to-one. More specifically, each component may access multiple files, and each file may be accessed by more than one component.

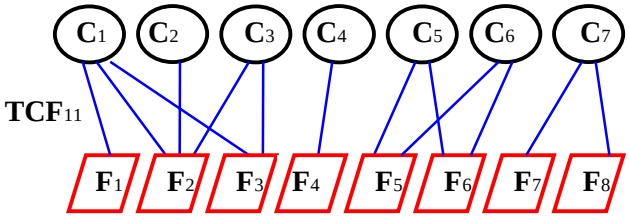

Figure 4. Storage access pattern of an application

Remark: The above application model can be used for simultaneous assignment of resources to multiple applications. A single big graph can be constructed with all the components from all the applications, where each application is represented by a sub-graph. Two sub-graphs are disconnected if the two corresponding applications do not communicate with each other. The same idea can be applied to the storage access graph.

To summarize, the application model contains the following sets and parameters:

Sets and indices

$c \in C$: Set of application components.

$f \in F$: Set of files to be placed on storage devices.

$l \in L$: Set of directed links in the application architecture graph.

$c' \in N_c$: Set of components that communicate with component *c*, i.e., $N_c = \{c' \in C : (c, c') \in L\}$.

Parameters:

T: $|C| \times |C|$-dim matrix. $T_{cc'}$ is the amount of traffic from component c to component c'.

TCF: $|C| \times |F|$-dim matrix. TCF_{cf} is the amount of write traffic from component c to file f.

TFC: $|F| \times |C|$-dim matrix. TFC_{fc} is the amount of read traffic from file f to component c.

TO: $|C|$-dim vector. $TO_c = \sum_{c' \in N_c} T_{cc'}$ is the total amount of LAN traffic originating from component *c*.

TI: $|C|$-dim vector. $TI_c = \sum_{c' \in N_c} T_{c'c}$ is the total amount of LAN traffic received by component *c*.

4. The Resource Model

This section describes the mathematical models for the processing, networking and storage resources in a computing utility. The collection of resources as a whole is referred to as the "utility fabric", which includes servers that can be assigned to applications, the Ethernet (LAN) fabric that connects the servers to each other, and the storage area network (SAN) fabric that connects the servers to the centralized storage devices.

4.1 Server attributes

Let *S* be the set of servers in the physical network. Similar to the application component model, a server's processing capability is characterized by a set of attributes. The value for each attribute may be fixed, or configurable. For example, a server may have a CPU speed of 550 MHZ, but its memory size is changeable between 4 and 8 MB. For each server $s \in S$, we use the set V_{sp} to represent its possible values for attribute $p \in P$. Note that the notion of a "server" here is not restricted to a compute server. It can be a firewall, a load balancer, a network attached storage (NAS) device, a VPN gateway, or any other device an application may need as a component. A server attribute "server type" can be used to distinguish between different kinds of servers.

4.2 Common networking assumptions

Before describing the mathematical models for the networking fabric, we first present a common set of networking assumptions we made to simply the models.

- We assume that all the network links are duplex links and traffic can flow in either direction. In addition, link capacities for the two directions can be different.
- For any physical link in any direction, its "link capacity" is indeed the minimum of the bandwidth capacities of the link, the source port and the destination port.

- Multiple physical links between two devices that are all active and load balanced are combined into one logical link with aggregated capacity. For example, four 1 Gbit/sec physical links can be combined to form one 4 Gbit/sec link in the logical topology. This simplification is valid when the combined links have equal bandwidth and share approximately equal load, which is typically true. It also comes naturally if trunking technology is applied on the links [5].
- If two switches appear in a redundant pair to avoid single point of failure, then redundant paths exist between at least one pair of devices in the physical topology. This can be simplified in different ways depending on the network protocol the switches implement. For example, in the LAN fabric, the spanning tree protocol [15] may be enforced, resulting in all the redundant paths between two network devices being blocked except one. If two switches in a redundant pair are both active and being load balanced, then we can partition the switches or servers that are connected to these two switches into two sets, one under each switch. And the cross links will be blocked. On the other hand, the SAN fabric may implement the Fabric Shortest Path First (FSPF) protocol [5], which assures uniform traffic load sharing over equivalent paths. Moreover, the two links in the same segment of the two paths usually have the same bandwidth. As a consequence, we can merge a pair of redundant switches into one switch. Corresponding links will also be merged to form a bigger link with aggregated bandwidth.

These simplifying assumptions are applied to both the LAN and the SAN fabrics as they are represented using mathematical models, and will not be repeated later on.

4.3 The LAN fabric

We assume that the logical topology of the LAN fabric in the computing utility is a tree. This is a reasonable assumption given that a layer-two switched network often implements the spanning tree protocol [15], guaranteeing that there is one and only one active path between two network devices. The tree network topology significantly simplifies the formulation of our problem later on.

Figure 5 shows an example of the LAN fabric topology. At the top is a switching/routing device that connects the utility fabric to the Internet or other utility fabrics. It is referred to as a root switch. Below the root switch is a set of edge switches, and below the edge switches is a set of rack switches. Servers are directly connected to either an edge switch or a rack switch. As the figure shows, an edge switch can be connected to a set of rack switches, a set of servers, or a combination of both. Note that this topology is more general than the one studied in [17,18,24], where all the servers connect to the rack switches only. The three-layer network shown here is chosen for demonstration purpose. It is straightforward to apply the methodology in this paper to a tree network with fewer or more layers.

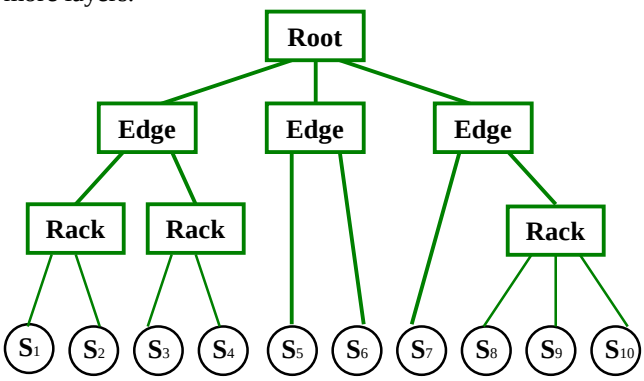

Figure 5. An example of the LAN fabric tree topology

The mathematical model contains the following sets and parameters:

Sets and Indices

$s \in S$: Set of servers.

$r \in R$: Set of rack switches.

$e \in E$: Set of edge switches.

$R_e \subset R$: Set of rack switches connected to edge switch *e*.

$SR_r \subset S$: Set of servers connected to rack switch *r*.

$SE_e \subset S$: Set of servers connected (directly or indirectly) under edge switch *e*.

Parameters:

BSI_s : The incoming bandwidth of server *s*.

BSO_s : The outgoing bandwidth of server *s*.

BRI_r : The incoming bandwidth of rack switch *r*.

BRO_r : The outgoing bandwidth of rack switch *r*.

BEI_e : The incoming bandwidth of edge switch *e*.

BEO_e : The outgoing bandwidth of edge switch *e*.

For easy indexing, each logical link in the network is associated with the device it can be uniquely identified with. For example, the link that connects server *s* to a rack or edge switch is associated with that server and its downstream/upstream bandwidth is referred to as the incoming/outgoing bandwidth of server *s*. The same rule applies to the links at the upper layers.

4.4 The SAN fabric

Various SAN topologies have been used in practice. The popular ones include ring, cascade, mesh, and core/edge topologies. Among these, the core/edge topology provides better resiliency, scalability, flexibility and throughput [5], and is adopted by many vendors and SAN designers. Therefore, we assume that the SAN fabric

in a computing utility has a core/edge topology. Figure 6 exemplifies a SAN with this topology.

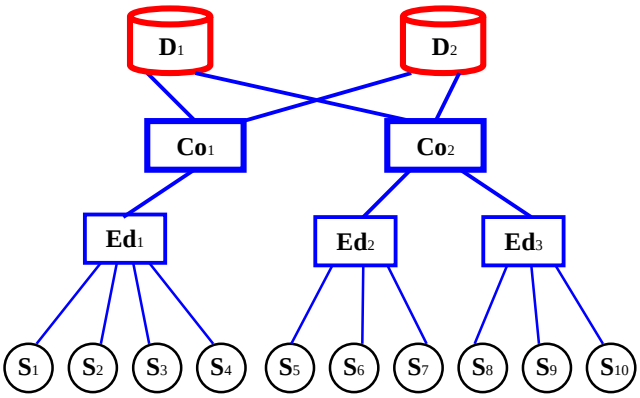

Figure 6. An example of the core/edge SAN topology

The core/edge topology contains two layers of switches. The core layer consists of at least one pair of redundant switches that are typically the most powerful. All the other switches connected to the core switches are referred to as edge switches. The centralized storage devices, such as disk arrays, are attached directly to the core switches, and the servers are attached directly to the edge switches. The above topology ensures that every storage device is accessible by any server in the SAN. Note that this logical topology is a simplification from the physical topology with redundancies in network devices and links.

It is worth pointing out that the servers in the this figure are exactly the same as those in Figure 5. The LAN network in Figure 5 carries communication traffic between these servers, and the SAN network in Figure 6 carries I/O traffic between the servers and the storage devices.

The mathematical model for the SAN contains the following sets and parameters.

Sets and indices:

$s \in S$: Set of servers.

$d \in D$: Set of storage devices.

$k \in K$: Set of FC core switches in the SAN.

$g \in G$: Set of FC edge switches in the SAN.

$SED_g \subset S$ **:** Set of servers connected to FC edge switch g.

$SCO_k \subset S$: Set of servers (indirectly) connected to FC core switch k.

Parameters:

BDC: $|D| \times |K|$-dim matrix. BDC_{dk} is the bandwidth of the FC link going from storage device d to core switch k.

BCD: $|K| \times |D|$-dim matrix. BCD_{kd} is the bandwidth of the FC link going from core switch k to storage device d.

BCE: $|G|$-dim vector. BCE_g is the bandwidth of the FC link going from a core switch to edge switch g.

BEC: $|G|$-dim vector. BEC_g is the bandwidth of the FC link going from edge switch g to a core switch.

BES: $|S|$-dim vector. BES_s is the bandwidth of the FC link going from an edge switch to server s.

BSE: $|S|$-dim vector. BSE_s is the bandwidth of the FC link going from server s to an edge switch.

Note that the complete topology of the utility fabric should be a combination of Figure 5 and Figure 6. It will not be shown here.

5. The Optimization Problem

This section describes the formulation of the mathematical optimization problem associated with RAP.

5.1 The decision variable

The resource assignment problem in this paper concerns selecting the right server in the utility fabric for each application component, represented by the following matrix of binary variables: For all $c \in C$ and $s \in S$,

$$x_{cs} = \begin{cases} 1 & \text{server s assigned to component c;} \\ 0 & \text{otherwise.} \end{cases}$$

In addition, the following two matrices of binary variables are defined. For all $c \in C$, $r \in R$, and $e \in E$,

$$zr_{cr} = \begin{cases} 1 & \text{rack switch r assigned to component c;} \\ 0 & \text{otherwise.} \end{cases}$$

$$ze_{ce} = \begin{cases} 1 & \text{edge switch e assigned to component c;} \\ 0 & \text{otherwise.} \end{cases}$$

Here we say a switch is assigned to a component if at least one server connected (directly or indirectly) under the switch is assigned to that component. Note that these two variables are redundant to the variables x_{cs}. They are only introduced to help express the Ethernet bandwidth constraints in a more succinct way, and to make solving of the problem more efficient, as shown in later sections.

5.2 The objective function

Resources in a computing utility can be assigned to application components based on many criteria, such as application performance, resource utilization, operator policies, or economic concerns. These can be associated with different objective functions of the optimization problem. In this paper, we choose the objective function used in the node placement optimization problem [4], which minimizes the traffic-weighted average inter-server distance where distance is measured in terms of network hop count. Let $DIST_{ss'}$ be the distance between two

servers *s* and *s'*, and $TSS_{ss'}$ be the amount of LAN traffic from server *s* to server *s'* as a result of server assignment. Then the objective function is

$$\text{Min} \quad J1 = \sum_{s,s' \in S} DIST_{ss'} * TSS_{ss'} .$$

As we can see, $TSS_{ss'} = \sum_{c \in C} \sum_{c' \in N_c} x_{cs} T_{cc'} x_{c's'}$. The value of $DIST_{ss'}$ depends on the relative location of server s and s'. For example, $DIST_{ss'} = 2$ if both servers are directly connected to the same switch, which is a preferred situation if these two servers communicate heavily.

By dividing the set of all server pairs into a number of subsets, each with a different $DIST_{ss'}$ value, then calculating the summation on each subset and adding them up, we get

$$J1 = 2\sum_{c \in C} (TO_c + TI_c) + \sum_{r \in R} \sum_{c \in C} zr_{cr} (TO_c + TI_c)$$
$$- 2\sum_{r \in R} \sum_{c \in C} \sum_{c' \in N_c} zr_{cr} T_{cc'} zr_{c'r} - \sum_{e \in E} \sum_{c \in C} \sum_{c' \in Nc} 2ze_{ce} T_{cc'} ze_{c'e}$$

The first term is the total amount of traffic originated from and received by all the components, which is a constant. Therefore, an equivalent objective function follows:

$$\text{Min} \quad J2 = \sum_{r \in R} \sum_{c \in C} zr_{cr} (TO_c + TI_c)$$
$$- 2\sum_{r \in R} \sum_{c \in C} \sum_{c' \in N_c} zr_{cr} T_{cc'} zr_{c'r} - \sum_{e \in E} \sum_{c \in C} \sum_{c' \in Nc} 2ze_{ce} T_{cc'} ze_{c'e}$$

This is a quadratic function of the binary variables zr_{cr} and ze_{ce}. The first term represents the total amount of traffic originated and received under all the rack switches. A similar term for all the edge switches, $\sum_{e \in E} \sum_{c \in C} ze_{ce} (TO_c + TI_c)$, would have been present, but was removed as part of the constant term. The second and third terms together capture the total amount of intra-switch traffic at all the switches. Here we define "intra-switch traffic" as the traffic flows whose source and destination nodes are servers under the same switch. The intuition is, as components that communicate heavily are placed close to each other in the network, the amount of intra-switch traffic is increased, which in turn results in smaller value for the objective function. In general, this leads to lower communication delay between application components inside the LAN fabric.

We choose not to include SAN latency in the objective function for the following two reasons. First, the SAN topology in our problem has the property that the number of hops for each data flow is fixed at 3 because any server and storage device pair is connected through two FC switches. This means, any server assignment solution results in the same SAN latency measure. Second, storage systems latency is dominated by I/O access at the storage device, which is typically several orders of magnitude larger than the SAN latency. Therefore, even if we could reduce the number of hops between a server and a storage device, it is inconsequential with respect to storage access latency. On the other hand, link capacity in the SAN is usually a concern in storage systems performance. Given the high cost of SAN switches, grossly over-provisioning may not be preferred, while at the same time we do not want the SAN fabric to be easily saturated. With this observation, we choose to deal with SAN link capacity in RAP without adding any new objective function. The rest of this section describes constraints in the problem that limit the search space for optimal server assignment solutions.

5.3 Preprocessing server feasibility

Before we start describing constraints in our problem, we need to define a server feasibility matrix *FS*, where

$$FS_{cs} = \begin{cases} 1 & \text{switch s meets the processing, networking,} \\ & \text{and I/O requirements of component c;} \\ 0 & \text{otherwise.} \end{cases}$$

More specifically, $FS_{cs} = 1$ if and only if

(a) $V_{sp} \cap VREQ_{cp} \neq \phi, \quad \forall p \in P$,

(b) $\sum_{c' \in N_c} T_{c'c} \leq BSI_s$ and $\sum_{c' \in N_c} T_{cc'} \leq BSO_s$,

(c) $\sum_{f \in F} TCF_{cf} \leq BSE_s$ and $\sum_{f \in F} TFC_{cf} \leq BES_s$.

Condition (a) ensures that server *s* matches the server attribute requirement by component *c*. Condition (b) ensures that the aggregate LAN traffic at each component *c* does not exceed the link bandwidth of server *s* in either direction. And condition (c) guarantees that the total amount of SAN traffic at each component *c* does not exceed the I/O bandwidth of server *s* in either direction.

The server feasibility matrix can be pre-computed before the optimization problem is solved. When the matrix *FS* is sparse, the search space for the optimization problem can be significantly reduced.

In the same spirit, we can define feasibility matrices *FR* and *FE* for rack and edge switches, respectively, where $FR_{cr} = 1$ if there is at least one feasible server under rack switch *r* for component *c*, $FE_{ce} = 1$ if there is at least one feasible server under edge switch *e* for component *c*. These two matrices can also be pre-computed.

5.4 The constraints

The constraints on the decision variables are as follows.

Normality constraints:

1) One and only one server is assigned to each application component.

$$\sum_{s\in S} x_{cs} = 1, \quad \forall c \in C$$

2) Each server can be assigned to at most one component.[1]

$$\sum_{c\in C} x_{cs} \le 1, \quad \forall s \in S$$

Variable relationship constraints:

3) A rack switch is assigned to a component if and only if a server under this rack switch is assigned to this component.

$$\sum_{s\in SR_r} x_{cs} = zr_{cr}, \quad \forall c \in C,\ r \in R$$

4) An edge switch is assigned to a component if and only if a server under this edge switch is assigned to this component.

$$\sum_{s\in SE_e} x_{cs} = ze_{ce}, \quad \forall c \in C,\ e \in E$$

LAN fabric constraints:

5) The LAN traffic going out of each rack switch to an edge switch does not exceed the link capacity.

$$\sum_{c\in C} TO_c zr_{cr} - \sum_{c\in C}\sum_{c'\in N_c} zr_{cr} T_{cc'} zr_{c'r} \le BRO_r, \quad \forall r \in R$$

Remember that TO_c is the total amount of LAN traffic originating from component *c*. On the left hand side, the first item represents the total amount of traffic originating under rack switch *r*, and the second item represents the amount of intra-switch traffic at this switch. Hence, the left hand side represents the amount of traffic passing through switch *r*, which should be bounded by the outgoing link bandwidth at the switch.

The derivation of the following three constraints is similar, therefore will be omitted.

6) The LAN traffic coming into each rack switch from an edge switch does not exceed the link capacity.

$$\sum_{c\in C} TI_c zr_{cr} - \sum_{c\in C}\sum_{c'\in N_c} zr_{cr} T_{cc'} zr_{c'r} \le BRI_r, \quad \forall r \in R$$

Remember that TI_c is the total amount of LAN traffic received by component *c*.

7) The LAN traffic going out of each edge switch to the root switch does not exceed the link capacity.

$$\sum_{c\in C} TO_c ze_{c,e} - \sum_{c\in C}\sum_{c'\in Nc} ze_{ce} T_{cc'} ze_{c'e} \le BEO_e, \quad \forall e \in E$$

8) The LAN traffic coming into each edge switch from the root switch does not exceed the link capacity.

$$\sum_{c\in C} TI_c ze_{ce} - \sum_{c\in C}\sum_{c'\in Nc} ze_{ce} T_{cc'} ze_{c'e} \le BEI_e, \quad \forall e \in E$$

SAN fabric constraints:

9) The SAN traffic going out of each FC edge switch to a core switch does not exceed the link capacity.

$$\sum_{s\in SED_g}\sum_{f\in F}\sum_{c\in C} TCF_{cf} x_{cs} \le BEC_g, \quad \forall g \in G$$

10) The SAN traffic coming into each FC edge switch from a core switch does not exceed the link capacity.

$$\sum_{s\in SED_g}\sum_{f\in F}\sum_{c\in C} TFC_{fc} x_{cs} \le BCE_g, \quad \forall g \in G$$

11) The SAN traffic from an FC core switch to a storage device does not exceed the link capacity.

$$\sum_{s\in SCO_k}\sum_{f\in F}\sum_{c\in C} TCF_{cf} x_{cs} Y_{fd} \le BCD_{kd}, \quad \forall k \in K,\ d \in D$$

Here Y_{fd} is a binary parameter, where $Y_{fd} = 1$ if and only if file *f* is placed on storage device *d*. As was discussed in Section 2, we choose to separate the file placement problem from the server assignment problem. The former has Y_{fd} as its decision variable. The solution is fed into our RAP problem as an input.

12) The SAN traffic from a storage device to an FC core switch does not exceed the link capacity.

$$\sum_{s\in SCO_k}\sum_{f\in F}\sum_{c\in C} TFC_{fc} x_{cs} Y_{fd} \le BDC_{dk}, \quad \forall k \in K,\ d \in D$$

Feasibility constraints:

13) All the variables are binary, and all the assigned servers, rack switches, and edge switches are feasible.

$$x_{cs} \in \{0, FS_{cs}\}, zr_{cr} \in \{0, FR_{cr}\}, ze_{ce} \in \{0, FE_{ce}\}$$

5.5 Summary

In summary, the complete formulation of the optimization problem for RAP follows.

$$\text{Min} \quad J2 = \sum_{r\in R}\sum_{c\in C} zr_{cr}(TO_c + TI_c)$$
$$- 2\sum_{r\in R}\sum_{c\in C}\sum_{c'\in N_c} zr_{cr} T_{cc'} zr_{c'r} - \sum_{e\in E}\sum_{c\in C}\sum_{c'\in Nc} 2ze_{ce} T_{cc'} ze_{c'e}$$

s.t.
$$\sum_{s\in S} x_{cs} = 1, \quad \forall c \in C \qquad (1)$$
$$\sum_{c\in C} x_{cs} \le 1, \quad \forall s \in S \qquad (2)$$
$$\sum_{s\in SR_r} x_{cs} = zr_{cr}, \quad \forall c \in C,\ r \in R \quad (3)$$
$$\sum_{s\in SE_e} x_{cs} = ze_{ce}, \quad \forall c \in C,\ e \in E \quad (4)$$
$$\sum_{c\in C} TO_c zr_{cr} - \sum_{c\in C}\sum_{c'\in N_c} zr_{cr} T_{cc'} zr_{c'r} \le BRO_r, \ \forall r \in R \quad (5)$$
$$\sum_{c\in C} TI_c zr_{cr} - \sum_{c\in C}\sum_{c'\in N_c} zr_{cr} T_{cc'} zr_{c'r} \le BRI_r, \ \forall r \in R \quad (6)$$

[1] To allow a server to be shared by multiple application components, constraint 2) can be replaced by a constraint on the total amount of processing capacity required by all the components co-locating on that server.

$$\sum_{c\in C} TO_c ze_{c,e} - \sum_{c\in C}\sum_{c'\in Nc} ze_{ce} T_{cc'} ze_{c'e} \le BEO_e, \forall e \in E \quad (7)$$

$$\sum_{c\in C} TI_c ze_{ce} - \sum_{c\in C}\sum_{c'\in Nc} ze_{ce} T_{cc'} ze_{c'e} \le BEI_e, \ \forall e \in E \quad (8)$$

$$\sum_{s\in SED_g}\sum_{f\in F}\sum_{c\in C} TCF_{cf}\, x_{cs} \le BEC_g, \quad \forall g \in G \quad (9)$$

$$\sum_{s\in SED_g}\sum_{f\in F}\sum_{c\in C} TFC_{fc}\, x_{cs} \le BCE_g, \quad \forall g \in G \quad (10)$$

$$\sum_{s\in SCO_k}\sum_{f\in F}\sum_{c\in C} TCF_{cf}\, x_{cs} Y_{fd} \le BCD_{kd}, \quad \forall k \in K, \ d \in D \quad (11)$$

$$\sum_{s\in SCO_k}\sum_{f\in F}\sum_{c\in C} TFC_{fc} x_{cs} Y_{fd} \le BDC_{dk}, \ \forall k \in K, \ d \in D \quad (12)$$

$$x_{cs} \in \{0, FS_{cs}\}, \quad \forall c \in C, s \in S$$

$$zr_{cr} \in \{0, FR_{cr}\}, ze_{ce} \in \{0, FE_{ce}\}, \quad \forall c \in C, r \in R, e \in E$$

The above is a nonlinear combinatorial optimization problem, which has been proven as NP-hard [19]. This problem is referred to as the original formulation of RAP and labeled as RAP0.

The problem formulation we described above can be applied to a number of different use cases.

a. Green-field assignment: This occurs when the first application is initially deployed in an empty utility.

b. Subsequent assignment: This occurs when there are existing applications running in the utility, and resources are assigned to the next application. In this case, the same application and resource models can be used, except that parameters in the resource model should reflect the remaining resource capacity.

c. Multiple applications assignment: This occurs when resources need to be assigned to more than one application at the same time. As discussed in Section 3, a larger application model with components from multiple applications can be used for this purpose.

d. **Dynamic assignment:** This occurs when an existing application requests for more resources as its real time workload intensity changes. In this case, a new application model will be submitted containing the additional requirement. Then depending on the application's ability to accommodate server migration, we can resolve the problem with or without fixing the existing server assignment.

e. **Automatic fail over**: This occurs when a server without high-availability configuration fails and needs replacement. We can find the best server to use from the pool of available servers using a similar RAP formulation.

The first three use cases only happen at application deployment time, while the last two use cases are useful at run time. Therefore, the former is at a time scale of days or longer, while the latter may be at a shorter time scale of minutes or hours.

The number of binary variables in RAP0 is $|C|\times(|S|+|R|+|E|)$, which is dominated by $|C|\times|S|$, the number of application components times the number of servers in the utility. It is conceivable that the problem becomes computationally more challenging as the infrastructure size or application size grows. Any heuristic search algorithms are not *guaranteed* to find a feasible and optimal solution. The next section presents three linearized formulations as mixed integer programming problems, which can be solved directly using a commercial solver, such as *CPLEX*.

6. Three Linearized Models

6.1 RAP-LINI

As we can see from the previous section, the original formulation RAP0 is nonlinear because the objective function and the LAN fabric constraints (5)-(8) are quadratic in binary variables zr_{cr} and ze_{ce}. This type of nonlinearity can be removed using a standard substitution technique with the observation that the product of binary variables is also binary. Let us define the following set of binary variables, $yr_{cc'r} = zr_{cr} zr_{c'r}$ and $ye_{cc'e} = ze_{ce} ze_{c'e}$, for all $c, c' \in C$, $r \in R$, $e \in E$.

With these new variables, the objective function can be rewritten as

$$\text{Min} \quad J2 = \sum_{r\in R}\sum_{c\in C} zr_{cr}(TO_c + TI_c) - 2\sum_{r\in R}\sum_{c\in C}\sum_{c'\in Nc} T_{cc'} yr_{cc'r} - 2\sum_{e\in E}\sum_{c\in C}\sum_{c'\in Nc} T_{cc'} ye_{cc'e}.$$

This is a linear combination of all the zr_{cr}, $yr_{cc'r}$ and $ye_{cc'e}$ variables. Similarly, constraints (5) and (8) in RAP0 can be rewritten as linear constraints as follows:

$$\sum_{c\in C} TO_c zr_{cr} - \sum_{c\in C}\sum_{c'\in Nc} T_{cc'} yr_{cc'r} \le BRO_r, \quad \forall r \in R \quad (5I)$$

$$\sum_{c\in C} TI_c zr_{cr} - \sum_{c\in C}\sum_{c'\in Nc} T_{cc'} yr_{cc'r} \le BRI_r, \quad \forall r \in R \quad (6I)$$

$$\sum_{c\in C} TO_c ze_{ce} - \sum_{c\in C}\sum_{c'\in Nc} T_{cc'} ye_{cc'e} \le BEO_e, \quad \forall e \in E \quad (7I)$$

$$\sum_{c\in C} TI_c ze_{ce} - \sum_{c\in C}\sum_{c'\in Nc} T_{cc'} ye_{cc'e} \le BEI_e, \quad \forall e \in E \quad (8I)$$

Additional constraints are needed to ensure that the $yr_{cc'r}$ variables behave as the product of binary variables. First, to ensure that $zr_{cr} = 0$ or $zr_{c'r} = 0 \Rightarrow yr_{cc'r} = 0$, we need

$$zr_{cr} \ge yr_{cc'r}, \quad zr_{c'r} \ge yr_{cc'r} \quad \forall c, c' \in C, r \in R. \quad (13I)$$

Second, to guarantee $zr_{cr} = 1$ and $zr_{c'r} = 1 \Rightarrow yr_{cc'r} = 1$, we would need

$$zr_{cr} + zr_{c'r} - yr_{cc'r} \le 1 \quad \forall c, c' \in C,\ r \in R.$$

However, since the objective function is to maximize a summation of the $yr_{cc'r}$ variables with non-negative coefficients, the second set of constraints are implied by the first set of constraints at optimality, and therefore are not required. Similarly, the following set of constraints should be imposed on the new $ye_{cc'e}$ variables.

$$ze_{ce} \ge ye_{cc'e}, \quad ze_{c'e} \ge ye_{cc'e} \quad \forall c, c' \in C, e \in E, \quad (14I)$$

Note that the new $yr_{cc'r}$ and $ye_{cc'e}$ variables only need to be continuous in the interval [0,1] instead of being binary. Let us take $yr_{cc'r}$ as an example. Based on the above discussion, constraint (13I) and the maximization nature of the objective function together ensure that $yr_{cc'r}$ behaves exactly as the product of zr_{cr} and $zr_{c'r}$. Since zr_{cr} and $zr_{c'r}$ are both binary, $yr_{cc'r}$ never really takes a fractional value between 0 and 1.

Remarks: The above substitution of variables results in a linear optimization problem with some integer variables and some continuous variables, thus a mixed integer programming problem. It is referred to as RAP-LINI, to be distinguished from the original nonlinear formulation RAP0. See Appendix I for the complete formulation of RAP-LINI. The main issue with this formulation is that the number of variables may be significantly higher than that of RAP0 with the introduction of $|C| \times |C| \times (|R| + |E|)$ continuous variables. There are a number of ways to improve the efficiency in solving the problem.

1. The number of $yr_{cc'r}$ and $ye_{cc'e}$ variables can be reduced in the following way:
   - $yr_{cc'r}$ is defined if and only if $FR_{cr} = 1, FR_{c'r} = 1, \text{and } T_{cc'} = 1$.
   - $ye_{cc'e}$ is defined if and only if $FE_{ce} = 1, FE_{c'e} = 1, \text{and } T_{cc'} = 1$.

   In all the other cases, the $yr_{cc'r}$ and $ye_{cc'e}$ variables are not needed in the formulation. This implies that, in the worst case where all the rack and edge switches are feasible for all the components, the number of extra variables in RAP-LINI is $|L| \times (|R| + |E|)$, i.e., the number of communication links in the application graph times the total number of LAN switches.
2. Since the number of zr_{cr} and ze_{ce} variables ($|C| \times (|R| + |E|)$) is usually significantly less than the number of x_{cs} variables $|C| \times |S|$, we can increase the efficiency of the branch and bound algorithm in the MIP solver by assigning higher priority to branching on variables ze_{ce} and zr_{cr}.

6.2 RAP-LINII

The previous subsection described a linearization technique that is straightforward and that resulted in a MIP formulation with $|L| \times (|R| + |E|)$ additional continuous variables than RAP0. This subsection describes a relatively more sophisticated linearization scheme, which leads to another MIP formulation with possibly fewer extra variables.

When looking at the LAN traffic flowing through each rack switch, we have the following intuition. For all $c \in C$ and $r \in R$, $zr_{cr}TO_c$ is the amount of traffic originating from component c under switch r, and $\sum_{c' \in N_c} zr_{c'r}T_{cc'}$ is the amount of traffic originating from component c and received under switch r. Now let us define a new variable,

$$tro_{cr} = zr_{cr}TO_c - zr_{cr} \sum_{c' \in N_c} zr_{c'r}T_{cc'},$$

which captures the amount of traffic that originated from component c under switch r and leaves switch r.

By definition of zr_{cr}, we have

$$tro_{cr} = \begin{cases} zr_{cr}TO_c - \sum_{c' \in N_c} zr_{c'r}T_{cc'}, & \text{if } zr_{cr} = 1; \\ 0, & \text{if } zr_{cr} = 0. \end{cases}$$

Therefore, we can equivalently define tro_{cr} as,

$$tro_{cr} = \max\left\{ zr_{cr}TO_c - \sum_{c' \in N_c} zr_{c'r}T_{cc'}, 0 \right\}.$$

Since tro_{cr} represents the amount of outgoing traffic from component c that passes through rack switch r, and the objective function tends to reduce the amount of traffic that passes through switches, the above definition can be enforced using the following two linear constraints:

$$tro_{cr} \ge zr_{cr}TO_c - \sum_{c' \in N_c} zr_{c'r}T_{cc'} \text{ and } tro_{cr} \ge 0. \quad (13II)$$

That is, these constraints will be binding at optimality.

Using the new variables tro_{cr}, the rack switch outgoing bandwidth constraint (5) in RAP0, $\sum_{c \in C} (zr_{cr}TO_c - zr_{cr} \sum_{c' \in N_c} zr_{c'r}T_{cc'}) \le BRO_r$, can be rewritten as

$$\sum_{c \in C} tro_{cr} \le BRO_r, \quad \forall r \in R. \quad (5II)$$

Similarly, we can represent the amount of LAN traffic originating from component c that leaves edge switch e using the following new variable:

$$teo_{ce} = ze_{ce}TO_c - ze_{ce} \sum_{c' \in N_c} ze_{c'e}T_{cc'}.$$

This would be enforced by the following constraints:

$$teo_{ce} \geq ze_{ce} TO_c - \sum_{c' \in N_c} ze_{c'e} T_{cc'} \text{ and } teo_{ce} \geq 0 . \quad (15II)$$

Then constraint (7) of RAP0 can be rewritten as

$$\sum_{c \in C} teo_{ce} \leq BEO_e, \quad \forall e \in E . \quad (7II)$$

Analogous variables tri_{cr} (tei_{ce}) representing the amount of incoming traffic to component *c* under rack switch *r* (edge switch *e)* from components outside the switch can be defined, with the following additional constraints:

$$tri_{cr} \geq zr_{cr} TI_c - \sum_{c' \in N_c} zr_{c'r} T_{c'c} \text{ and } tri_{cr} \geq 0 \quad (14II)$$

$$tei_{ce} \geq ze_{ce} TI_c - \sum_{c' \in N_c} ze_{c'e} T_{c'c} \text{ and } tei_{c,e} \geq 0 \quad (16II)$$

Then constraints (6) and (8) of RAP0 can be rewritten as

$$\sum_{c \in C} tri_{cr} \leq BRI_r, \quad \forall r \in R . \quad (6II)$$

$$\sum_{c \in C} tei_{ce} \leq BRI_e, \quad \forall e \in E . \quad (8II)$$

By comparing the definition of the new variables with the objective function J2 in RAP0, it is easy to see that,

$$J2 = \sum_{r \in R} \sum_{c \in C} (tro_{cr} + tri_{cr}) + \sum_{e \in E} \sum_{c \in C} (teo_{ce} + tei_{ce}) - \sum_{e \in E} \sum_{c \in C} ze_{ce} (TO_c + TI_c)$$

Since $\sum_{e \in E} \sum_{c \in C} ze_{ce} (TO_c + TI_c) = \sum_{c \in C} (TO_c + TI_c)$ is a constant, an equivalent objective function is the following.

$$\text{Min} \quad J3 = \sum_{r \in R} \sum_{c \in C} (tro_{cr} + tri_{cr}) + \sum_{e \in E} \sum_{c \in C} (teo_{ce} + tei_{ce})$$

The interpretation of the objective function follows. To minimize the traffic-weighted average inter-server distance, it is equivalent to minimize the total amount of traffic flowing on all the Ethernet links. Because the total amount of traffic originating from and received by all the application components is a constant, the total amount of traffic flowing on all the server-to-switch links is a constant. Therefore, an equivalent objective function is to minimize the total amount of inter-switch traffic, which is exactly what J3 is. The term "inter-switch traffic" refers to the traffic flowing on a link that connects two switches. These links are typically more expensive. And they are more likely to get saturated because they are often shared by multiple components, or even multiple applications. By minimizing the utilization of these shared links by a single application, we reduce the likelihood of creating bottlenecks in the LAN fabric.

Remarks: This MIP formulation of the resource assignment problem is referred to as RAP-LINII. Its complete formulation is shown in Appendix II. In this case, a total number of $2|C| \times (|R| + |E|)$ new continuous variables are introduced. This approach involves fewer extra variables than the RAP-LINI approach if $2|C| < |L|$, i.e., if each application component has, on average, more than 2 incident links.

6.3 RAP-MCFM

This section presents the same resource assignment problem formulated using a multicommodity flow model (MCFM) [1]. The approach is quite different from the previous two formulations. A multicommodity flow model contains multiple commodities, each going from its source to its sink in the form of network flows. The flows have to obey the flow conservation law at each node for each commodity. The capacity of each network link is shared by the flows it carries, possibly from multiple commodities, and the capacity limit is enforced.

We need to define two new sets and a new parameter.

- $n \in N$: Set of all nodes in the LAN fabric topology, including servers, rack switches and edge switches, i.e., $N = S \cup R \cup E$.
- $a \in A$: Set of arcs in the LAN fabric topology. Each arc *a* is an ordered pair (*m,n*) of nodes in *N*.
- B_a or B_{mn}: Link bandwidth of arc $a = (m,n) \in A$.

As mentioned in Section 4, the order of nodes in an arc matters. For example, for *a=(m,n)* and *a'=(n,m)* where $m, n \in N$, the values of B_a and $B_{a'}$ could be different.

The application model stays the same. Although in this formulation, each ordered pair of application components (*c,c'*) for which $T_{c,c'} > 0$ represents a commodity. Thus, there is a commodity corresponding to each link $l = (c,c') \in L$ in the application architecture graph.

The binary server assignment decision variables x_{cs} are still needed. In addition, for each commodity $(c,c') \in L$ and each arc $(m,n) \in A$, we define the continuous nonnegative flow variables $w_{cc'mn}$ to represent the flow along arc $(m,n) \in A$, in the direction from *m* to *n*, that originated from component *c* and is destined to component c'.

The previous objective of minimizing the traffic-weighted average inter-server distance can be accomplished by minimizing the total amount of traffic on all the network arcs used by all the commodities:

$$\text{Min} \quad J1 = \sum_{(c,c') \in L} \sum_{(m,n) \in A} w_{cc'mn}$$

The constraints of RAP-MCFM are the following:

1)-2) Normality constraints (1) and (2) on the x_{cs} variables are still valid.

3) Link capacity constraints for all arcs $(m,n) \in A$ in the LAN.

$$\sum_{(c,c')\in L} w_{cc'mn} \le B_{mn}. \qquad (3III)$$

4) Flow conservation for each commodity $(c,c')\in L$ and each node $n\in N$.

a) For each switch $n\in R\cup E$, we have

$$\sum_{m\in N:(m,n)\in A} w_{cc'mn} = \sum_{p\in N:(n,p)\in A} w_{cc'np}. \qquad (4III)$$

b) For each server node $n\in S$,

$$\sum_{m\in N:(m,n)\in A} w_{cc'mn} - x_{c'n}T_{cc'} = \sum_{p\in N:(n,p)\in A} w_{cc'np} - x_{cn}T_{cc'}.$$

In general, the flow conservation law at a node requires that, for each commodity, the total amount of incoming flows at node *n*, minus the demand of the commodity at the node, should equal to the total mount of outgoing flows at node *n*, minus the supply of the commodity at the node. Since the switches do not produce or consume traffic, and they simply forward the traffic, the supply and demand terms become zero for the switch nodes.

For the server nodes, equation b) is conceptually correct. However, consider the follow fact. Since a server cannot be assigned to more than one component, for each commodity $(c,c')\in L$, a server cannot be the source and the sink at the same time. This means, in equation 4b), either the left-hand-side or the right-hand-side is zero for any commodity. Therefore, we can get tighter constraints by splitting equation b) into the following two equations.

i) Total amount of incoming flows equals demand:

$$\sum_{m\in N:(m,s)\in A} w_{cc'ms} = x_{c's}T_{cc'}. \qquad (5III)$$

ii) Total amount of outgoing flows equals supply:

$$\sum_{p\in N:(s,p)\in A} w_{cc'sp} = x_{cs}T_{cc'}. \qquad (6III)$$

5) SAN link capacity constraints (9)-(12) in RAP0 remain the same as long as the core/edge topology assumption on the SAN is valid. If we allow the SAN to have an arbitrary topology, the similar idea of using MCFM can be applied.

6) A feasibility matrix *FN* between components and nodes can be obtained the same way we computed *FS*, *FR* and *FE* for RAP0. For each component $c\in C$ and each node $n\in N$, $FN_{cn}=1$ means node *n* is feasible for component *c*. Note that the node can be a server, a rack switch or an edge switch. Then, the following feasibility constraints hold.

a) For the server assignment variables:

$$x_{cs}\in\{0, FN_{cs}\}, \quad \forall c\in C,\ \forall s\in S$$

b) For the flow variables $w_{cc'mn}$:

$$0\le w_{cc'np}\le T_{cc'}FN_{cn}, \quad \forall (c,c')\in L,\ \forall n\in N,(n,p)\in A$$

$$0\le w_{cc'mn}\le T_{cc'}FN_{c'n}, \quad \forall (c,c')\in L,\ \forall n\in N,(m,n)\in A$$

Remarks: This model is referred to as RAP-MCFM. It again contains a linear MIP problem. The complete formulation of RAP-MCFM is in Appendix III. The number of binary variables in this model is in the order of $|C|\times|S|$. And the number of continuous flow variables is in the order of $|L|\times|A|$, the total number of links in the application architecture times the total number of arcs in the LAN fabric topology. If the LAN topology is a tree, the number of arcs |A| is simply $2(|N|-1)$, where $|N|=|S|+|R|+|E|$. Compared to RAP-LINI and RAP-LINII, RAP-MCFM involves many more continuous variables and slightly fewer binary variables. The complexity comparison of the three linearized models in terms of number of variables is summarized in Table 1.

Table 1. Comparison of three linearized models

Model	RAP-LINI	RAP-LINII	RAP-MCFM
#Bin.	$\|C\|\times\|N\|$	$\|C\|\times\|N\|$	$\|C\|\times\|S\|$
#Cont.	$\|L\|\times(\|R\|+\|E\|)$	$2\|C\|\times(\|R\|+\|E\|)$	$\|L\|\times\|A\|$

When the topology of the LAN fabric is a tree, there is a unique path between every pair of servers. This means the flows will not be split, i.e., $w_{cc'mn}$ will be either zero or $T_{c,c'}$. Among the three linearized models, the RAP-MCFM model may not be the most efficient in this case since it involves a much larger MIP problem. However, if we do allow flow splitting, the RAP-MCFM formulation could be applied to a generalized network topology such as an arbitrary graph, which cannot be handled by the first two linear models. Along with the fact that the application architecture is also an arbitrary graph, RAP-MCFM is a fairly general formulation of the resource assignment problem, which can be applied to more general utility computing environments such as the Grid.

7. Case Studies

This section presents computation results from the three MIP formulations on a set of numerical examples.

7.1 Example 1: A UDC with a 4-tier e-commerce application

The first example involves a utility data center and a multi-tiered Web application. The LAN fabric contains one edge switch, 2 rack switches, a total of 125 servers, with 64 of them connected to rack switches and 61 of them, including 4 firewalls, connected directly to the edge switch. The SAN fabric contains 44 FC edge switches, 7 FC core switches and 2 large disk arrays. The compute servers can be classified into 6 different types with varying levels of processing power. The application we

chose is an e-commerce application with 4 tiers: one firewall, 10 Web servers, 18 application servers and 2 database servers. The storage requirements of the application were not available to us. So we assumed that there are a total of 30 files that have been pre-allocated to the two disk arrays. The problem was solved using *CPLEX 7.5* on an HP J6000 UNIX workstation with dual processors of 552MHz.

The solver was able to find the optimal solution with all three MIP formulations. The optimal objective function (O.F.) values are consistent from the three models, as shown in Table 2. It also shows the solution time in each case and the complexity of each model in terms of number of variables (binary and continuous), number of equations, and number of non-zero coefficients in the equations. As we can see, for this particular example, the first two linear formulations have roughly the same complexity and used the same amount of solution time. The RAP-MCFM formulation involves a much larger MIP problem and required much more time to solve, which is consistent with our expectation.

Table 2. Solution time and complexity of three models for example 1

Model	RAP-LINI	RAP-LINII	RAP-MCFM
Solution Time	2.6 sec	2.6 sec	85 sec
# Binary	2197	2197	2073
# Continuous	4362	4220	121,427
# Equations	1584	990	116,268
# Coefficients	31,900	34,062	497,618
Optimal O.F.	-7,680	0	7,680
Optimal J3	0	0	0

7.2 Example 2: A hypothetical 500-server data center with a 19-component application

To test the performance of the RAP formulations with respect to their application to a larger computing utility, we also experimented with a hypothetical utility fabric with 500 servers. The LAN fabric contains 5 edge switches and 25 rack switches. The SAN fabric contains 20 FC edge switches, 2 FC core switches, and 2 disk arrays. The hypothetical application has 19 components and 20 files. All the parameters for both the utility fabric and the application were synthetically generated based on data from real data centers and applications.

Again optimal solutions were found using all three MIP formulations. Their respective performance and complexity on a problem instance are shown in Table 3. In this case, the RAP-LINII is the most efficient with minimum complexity and took the least amount of solution time. The RAP-MCFM model is again a much larger model and took more than 5 minutes to solve, although during this time only 99.5 seconds was spent on computation and the rest was spent on model generation. This means there is room to improve the performance by more efficient model handling.

Table 3. Solution time and complexity of three models for example 2

Model	RAP-LINI	RAP-LINII	RAP-MCFM
Solution Time	52 sec	12 sec	319 sec
# Binary	5490	5490	5015
# Continuous	12,631	11,211	196,156
# Equations	7288	3308	183,149
# Coefficients	100,786	96,371	822,096
Optimal O.F.	-59.4	6	71.4
Optimal J3	6	6	6

7.3 Numerical scalability test

To see how the solution times of the RAP formulations scale with the size of the utility infrastructure, we performed a numerical experiment where we used the same application as in example 2, scaled up the number of servers from 100, 200, 300, 400 to 500, and solved each problem instance with the three models. In all cases the *CPLEX* engine was able to find an optimal solution. The total solution time (in seconds) for each case is shown in Table 4.

Table 4. Solution time (sec) comparison on 5 examples

# servers	RAP-LINI	RAP-LINII	RAP-MCFM
100	10.1	9.5	247
200	10.6	10.1	259
300	11.3	10.4	273
400	30.8	10.9	289
500	52	12	319

As we can see the RAP-LINII formulation is the most scalable among the three, for which the solution time remains fairly constant throughout the examples. For the RAP-LINI model, the solution time is similar to that of RAP-LINII until the 400-server example, where we see a jump in solution time. The reason for this jump is that, the LP relaxation was able to find the integer optimum solution immediately for the first three examples, while the last two examples required a branch and bound routine to find the optimum solution to the MIP problem. The RAP-MCFM model took much longer to solve compared to other two. However, the growth in computation time as the problem becomes larger is fairly moderate.

Other examples of varying sizes were tested and the results are quite similar, therefore will not be presented here. Note that a computing utility with 500 servers is a reasonably large one. Even though the longest solution

time with RAP-MCFM is over 5 minutes, it is quite negligible compared to the lifetime of the application, assuming that we are primarily interested in continuously running applications, such as enterprise business applications. Although the time varying nature of the application workload may require re-assignment of computing resources from time to time, we expect such updates to be implemented at a time scale of at least an hour, not minutes.

8. Conclusions and Future Work

This paper presented a resource assignment problem for a computing utility as a mathematical optimization problem. The original nonlinear problem was transformed into three MIP formulations that can be solved directly using *CPLEX*. Their respective computation complexity was demonstrated through a number of examples. In all cases, the RAP-LINII formulation was the most efficient and the RAP-MCFM, a multicommodity flow formulation, required the most computation time. However, the first two formulations assume that the LAN fabric of the utility infrastructure has a tree topology, while the RAP-MCFM formulation allows the LAN topology to be an arbitrary graph. Along with the general component-graph based application architecture, the RAP-MCFM formulation is fairly generic and can be applied to resource assignment for distributed applications in any network environment.

The scalability test we presented in this paper is quite preliminary. As ongoing work, we are conducting numerical experiments involving other parameters and many more examples. At the same time, we are searching for opportunities to implement the resource assignment techniques we developed in a real system to validate the effectiveness of our approach.

Future work on this problem includes performing sensitivity analysis on how variation in parameter values impact the solution quality, investigating techniques for resource pool de-fragmentation such as re-assignment and server migration, exploring an integrated optimization framework for both server assignment and storage placement, as well as studying policy management related to resource assignment.

Acknowledgements

The authors would like to thank Simge Kucukyavuz for suggestions that improved the efficiency of the multi-commodity flow model, as well as Jerry Rolia and Marsha Duro for helpful discussions on this problem.

References

[1] R.K. Ahuja, T.I Magnanti and J.B. Orlin, *Network Flows*, Prentice Hall, Englewood Cliffs, New Jersy, 1993.

[2] E. Anderson, M. Kallahalla, S. Spence, R. Swaminathan, and Q. Wang, "Ergastulum: quickly finding near-optimal storage system designs," submitted to *ACM Transactions on Computer Systems.*

[3] G.C. Armour and E.S. Buffa, "A heuristic algorithm and simulation approach to relative location of facilities," *Management Science*, vol. 9, pp. 294-309, 1963.

[4] S. Banerjee, B. Mukherjee and D. Sarkar, "Heuristic algorithms for constructing optimized structures in linear multihop lightwave networks," *IEEE Transactions on Communications*, vol. 42, pp. 1811-1826, 1994.

[5] Brocade Communications Systems, "Designing next-generation fabrics with Brocade switches," white paper, 2001.

[6] F.J. Corbato, J.H. Saltzer, and C.T. Clingen, "MULTICS – the first seven years," *Proceedings of the AFIPS Spring Joint Computer Conference*, 1972.

[7] L.W. Dowdy and D.V. Foster, "Comparative models of the file assignment problem," *ACM Computer Sueveys*, vol. 14(2), pp. 287-313, 1982.

[8] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, Oxford, UK, 1979.

[9] *The Grid: Buleprint for a New Computing Infrastructure*, edited by Ian Foster and Carl Kesselman, July 1998.

[10] www.ilog.com

[11] International Business Machines Corporation, "Living in an on demand world," white paper, Oct. 2002.

[12] A. Kothari, S. Suri and Y. Zhou, "Bandwidth-Constrained allocation in Grid computing," *Proceedings of the 8th Workshop on Algorithms and Data Structures*, July 2003.

[13] D.l. Kreher and D.R. Stinson, "Combinatorial algorithms: generation, enumeration, and search," CRC Press, 1999.

[14] Microsoft Corporation, "Building dynamic data center," white paper, May 2003.

[15] R. Perlman, *Interconnections: Bridges, Routers, Switches, and Internetworking Protocols*, Addison-Wesley, 1999.

[16] J. Rolia, S. Singhal and R. Friedrich, "Adaptive Internet data centers," *Proceedings of SSGRR 2000 Computer and eBusiness Conference*, July-Aug. 2000.

[17] C.A. Santos, X. Zhu and H. Crowder, "A mathematical optimization approach for resource allocation in large scale data centers," *HP Labs Technical Report, HPL-2002-64*, March 11th, 2002.

[18] C.A. Santos and X. Zhu, "Testing scalability of the mathematical programming approach for resource

allocation in a computing utility," *HP Labs Technical Report, HPL-2003-1*, January 6th, 2003.
[19] Troy Shahoumian, "IDC layout is NP-hard," internal document, March 2002.
[20] J. Rolia, X. Zhu and M. Arlitt, "Resource access management for a utility hosting enterprise applications," in *Proceedings of the 8th IFIP/IEEE Symposium on Integrated Network Management (IM)*, March 2003.
[21] Sun Microsystems, "N1 - Introducing just-in-time computing," white paper, 2002.
[22] Vernon Turner, "Utility Data Center: HP's first proof point for service-centric computing," IDC white paper, November 2001.
[23] L.A. Wolsey, *Integer Programming*, Wiley, 1998.
[24] X. Zhu and S. Singhal, "Optimal resource assignment in Internet data centers," *Proceedings of the 9th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (Mascots)*, pp. 61-69, August 2001.

Appendix I.
Complete Formulation of RAP-LINI

$$\text{Min} \quad J2 = \sum_{r\in R}\sum_{c\in C} zr_{cr}\,(TO_c + TI_c) - 2\sum_{r\in R}\sum_{c\in C}\sum_{c'\in Nc} T_{cc'}\,yr_{cc'r} - 2\sum_{e\in E}\sum_{c\in C}\sum_{c'\in Nc} T_{cc'}\,ye_{cc'e}$$

$$\text{s.t.} \quad \sum_{s\in S} x_{cs} = 1, \quad \forall c \in C \qquad (1)$$

$$\sum_{c\in C} x_{cs} \le 1, \quad \forall s \in S \qquad (2)$$

$$\sum_{s\in SR_r} x_{cs} = zr_{cr}, \quad \forall c \in C,\ r \in R \qquad (3)$$

$$\sum_{s\in SE_e} x_{cs} = ze_{ce}, \quad \forall c \in C,\ e \in E \qquad (4)$$

$$\sum_{c\in C} TO_c\, zr_{cr} - \sum_{c\in C}\sum_{c'\in Nc} T_{cc'}\, yr_{cc'r} \le BRO_r, \quad \forall r \in R \qquad (5I)$$

$$\sum_{c\in C} TI_c\, zr_{cr} - \sum_{c\in C}\sum_{c'\in Nc} T_{cc'}\, yr_{cc'r} \le BRI_r, \quad \forall r \in R \qquad (6I)$$

$$\sum_{c\in C} TO_c\, ze_{ce} - \sum_{c\in C}\sum_{c'\in Nc} T_{cc'}\, ye_{cc'e} \le BEO_e, \quad \forall e \in E \qquad (7I)$$

$$\sum_{c\in C} TI_c\, ze_{ce} - \sum_{c\in C}\sum_{c'\in Nc} T_{cc'}\, ye_{cc'e} \le BEI_e, \quad \forall e \in E \qquad (8I)$$

$$\sum_{s\in SED_g}\sum_{f\in F}\sum_{c\in C} TCF_{cf}\, x_{cs} \le BEC_g, \quad \forall g \in G \qquad (9)$$

$$\sum_{s\in SED_g}\sum_{f\in F}\sum_{c\in C} TFC_{fc}\, x_{cs} \le BCE_g, \quad \forall g \in G \qquad (10)$$

$$\sum_{s\in SCO_k}\sum_{f\in F}\sum_{c\in C} TCF_{cf}\, x_{cs} Y_{fd} \le BCD_{kd}, \quad \forall k \in K,\ d \in D \qquad (11)$$

$$\sum_{s\in SCO_k}\sum_{f\in F}\sum_{c\in C} TFC_{fc}\, x_{cs} Y_{fd} \le BDC_{dk}, \quad \forall k \in K,\ d \in D \qquad (12)$$

$$zr_{cr} \ge yr_{cc'r}, \quad zr_{c'r} \ge yr_{cc'r}, \quad \forall c, c' \in C, r \in R \qquad (13I)$$

$$ze_{ce} \ge ye_{cc'e}, \quad ze_{c'e} \ge ye_{cc'e}, \quad \forall c, c' \in C, e \in E \qquad (14I)$$

$$x_{cs} \in \{0, FS_{cs}\}, \quad \forall c \in C, s \in S$$

$$zr_{cr} \in \{0, FR_{cr}\}, ze_{ce} \in \{0, FE_{ce}\}, \quad \forall c \in C, r \in R, e \in E$$

$$yr_{cc'r} \ge 0, \quad yr_{cc'e} \ge 0, \quad \forall c, c' \in C, r \in R, e \in E$$

Appendix II.
Complete Formulation of RAP-LINII

$$\text{Min } J3 = \sum_{r\in R}\sum_{c\in C}(tro_{cr} + tri_{cr}) + \sum_{e\in E}\sum_{c\in C}(teo_{ce} + tei_{ce})$$

$$\text{s.t.} \quad \sum_{s\in S} x_{cs} = 1, \quad \forall c \in C \qquad (1)$$

$$\sum_{c\in C} x_{cs} \le 1, \quad \forall s \in S \qquad (2)$$

$$\sum_{s\in SR_r} x_{cs} = zr_{cr}, \quad \forall c \in C,\ r \in R \quad (3)$$

$$\sum_{s\in SE_e} x_{cs} = ze_{ce}, \quad \forall c \in C,\ e \in E \quad (4)$$

$$\sum_{c\in C} tro_{cr} \le BRO_r, \quad \forall r \in R \qquad (5II)$$

$$\sum_{c\in C} tri_{cr} \le BRI_r, \quad \forall r \in R \qquad (6II)$$

$$\sum_{c\in C} teo_{ce} \le BEO_e, \quad \forall e \in E \qquad (7II)$$

$$\sum_{c\in C} tei_{ce} \le BRI_e, \quad \forall e \in E \qquad (8II)$$

$$\sum_{s\in SED_g}\sum_{f\in F}\sum_{c\in C} TCF_{cf}\, x_{cs} \le BEC_g, \quad \forall g \in G \qquad (9)$$

$$\sum_{s\in SED_g}\sum_{f\in F}\sum_{c\in C} TFC_{fc}\, x_{cs} \le BCE_g, \quad \forall g \in G \qquad (10)$$

$$\sum_{s\in SCO_k}\sum_{f\in F}\sum_{c\in C} TCF_{cf}\, x_{cs} Y_{fd} \le BCD_{kd}, \quad \forall k \in K,\ d \in D \qquad (11)$$

$$\sum_{s\in SCO_k}\sum_{f\in F}\sum_{c\in C} TFC_{fc}\, x_{cs} Y_{fd} \le BDC_{dk}, \quad \forall k \in K,\ d \in D \qquad (12)$$

$$zr_{cr} TO_c - \sum_{c'\in N_c} zr_{c'r} T_{cc'} \le tro_{cr}, \ \forall c \in C, r \in R \quad (13II)$$

$$zr_{cr} TI_c - \sum_{c'\in N_c} zr_{c'r} T_{c'c} \le tri_{cr}, \ \forall c \in C, r \in R \quad (14II)$$

$$ze_{ce} TO_c - \sum_{c'\in N_c} ze_{c'e} T_{cc'} \le teo_{ce}, \ \forall c \in C, e \in E \quad (15II)$$

$$ze_{ce} TI_c - \sum_{c'\in N_c} ze_{c'e} T_{c'c} \le tei_{ce}, \ \forall c \in C, e \in E \quad (16II)$$

$$x_{cs} \in \{0, FS_{cs}\}, \quad \forall c \in C, s \in S$$

$$zr_{cr} \in \{0, FR_{cr}\}, ze_{ce} \in \{0, FE_{ce}\}, \quad \forall c \in C, r \in R, e \in E$$

$$tro_{cr} \ge 0, tri_{cr} \ge 0, teo_{ce} \ge 0, tei_{ce} \ge 0, \forall c \in C, r \in R, e \in E$$

Appendix III.
Complete Formulation of RAP-MCFM

$$\text{Min} \quad J1 = \sum_{(c,c')\in L}\ \sum_{(m,n)\in A} w_{cc'mn}$$

$$\text{s.t.} \quad \sum_{s\in S} x_{cs} = 1, \quad \forall c \in C \qquad (1)$$

$$\sum_{c\in C} x_{cs} \le 1, \quad \forall s \in S \qquad (2)$$

$$\sum_{(c,c')\in L} w_{cc'mn} \le B_{mn}, \quad \forall (m,n) \in A \qquad (3III)$$

$$\sum_{m\in N:(m,n)\in A} w_{cc'mn} = \sum_{p\in N:(n,p)\in A} w_{cc'np}, \ \forall (c,c') \in L, n \in R \cup E \qquad (4III)$$

$$\sum_{m\in N:(m,s)\in A} w_{cc'ms} = x_{c's} T_{cc'}, \forall (c,c') \in L, s \in S \quad (5III)$$

$$\sum_{p\in N:(s,p)\in A} w_{cc'sp} = x_{cs} T_{cc'}, \forall (c,c') \in L, s \in S \quad (6III)$$

$$\sum_{s\in SED_g}\sum_{f\in F}\sum_{c\in C} TCF_{cf}\, x_{cs} \le BEC_g, \quad \forall g \in G \qquad (9)$$

$$\sum_{s\in SED_g}\sum_{f\in F}\sum_{c\in C} TFC_{fc}\, x_{cs} \le BCE_g, \quad \forall g \in G \qquad (10)$$

$$\sum_{s\in SCO_k}\sum_{f\in F}\sum_{c\in C} TCF_{cf}\, x_{cs} Y_{fd} \le BCD_{kd}, \quad \forall k \in K,\ d \in D \qquad (11)$$

$$\sum_{s\in SCO_k}\sum_{f\in F}\sum_{c\in C} TFC_{fc} x_{cs} Y_{fd} \le BDC_{dk}, \ \forall k \in K,\ d \in D \quad (12)$$

$$x_{cs} \in \{0, FS_{cs}\}, \quad \forall c \in C, s \in S$$

$$0 \le w_{cc'np} \le T_{cc'} FS_{cn}, \quad \forall (c,c') \in L, \ \forall n \in N, (n,p) \in A$$

$$0 \le w_{cc'mn} \le T_{cc'} FS_{c'n}, \quad \forall (c,c') \in L, \ \forall n \in N, (m,n) \in A$$