Published at 2011 31st International Conference on Distributed Computing Systems Workshops

DC-PET: A System for Data Center Performance Estimation

Peer Hasselmeyer, Nico d'Heureuse
NEC Laboratories Europe
Kurfürstenanlage 36
69115 Heidelberg, Germany
Hasselmeyer|dHeureuse@neclab.eu

Abstract

Designing data centers with adequate performance and at the right scale is a difficult task. The load in terms of network and CPU utilization is hard to estimate without appropriate tools. In this paper we introduce a data center performance estimation system which can support planning data centers and application deployments. The tool takes data center topologies and applications and calculates the load expected on the server and networking infrastructure. To quickly produce results, the data center evaluator uses linear programming.

1. Introduction

Data centers (DCs) are the backbone of the current information infrastructure. They not only process Web traffic, but also manage company-internal processes, satisfy storage needs, and facilitate communication and collaboration. Data centers are large and complex setups of compute, storage and network devices. With the advent of the cloud operating model, more and more storage and compute loads are aggregated in ever-larger data centers.

Building a data center includes many design decisions, including the selection of a network topology, the link bandwidth, and the server capacity. As data centers are large, complex systems, tool support is desired to guide the decisions. The majority of questions arising in the design phase are "what-if" kind of questions where DC designers want to quickly see what happens if some parameters change, e.g. the network topology or the application work load. For these kinds of questions, perfect accuracy is not as important as the time needed to get an answer. Rough estimates are enough to support the design process.

In order to make designing data centers easier and to facilitate their comparison, the authors are developing a data center planning and evaluation tool (called DC-PET). The tool allows assessing the impact of particular workloads on the data center infrastructure. Many parameters can be adjusted in order to make answering "what-if" questions possible. The tool is based on linear programming letting it produce answers quickly.

This paper introduces the capabilities of the DC-PET tool. The methods for modeling data centers and applications are detailed. The statistical metrics that the tool can calculate and display are described and some initial results are presented.

2. Data Center Modeling

For allowing DC-PET to analyze data center load, all relevant artifacts of the data center must be modeled and expressed in a machine-processable way. The model consists of two main abstractions: the physical hardware and the applications deployed on that hardware. In this section, the abstractions used to describe data center hardware are introduced.

The physical data center hardware contains devices and links between them. The devices perform various functions while the links are used to exchange information between the devices. The following device types are supported by DC-PET:

Switches: Switches connect multiple devices, such as servers and other switches. They relay traffic from one link to another.

Servers: Servers host and run (parts of) applications. Servers are general-purpose in the sense that they can host any type of application function. Depending on the applications running on the server, incoming traffic can result in further traffic originating at the server, i.e., a single incoming request can result in one or more outgoing chunks of data.

A data center might contain servers that are designed to perform only specific functions. They are therefore not general-purpose anymore and only application components performing a compatible function are allowed to be assigned to these dedicated servers. Examples are load balancers, firewalls, and storage servers. Within DC-PET, servers can be assigned to arbitrarily defined groups. Every server can be part of one or multiple groups. The deployment of application components can then be restricted to servers of certain groups.

Gateways: A gateway represents a connection of the data center to the outside world, which means, usually, "the Internet" or the data center backbone network. In our model we assume that all traffic in the data center is directly or indirectly created by requests entering the data center from the outside through a gateway. Traffic and application execution that is completely internal to the data center can be modeled by additional gateways that produce the triggers for the internal traffic.

The devices are connected via physical links which allow traffic to flow between them. Different topologies can be used to wire data centers. DC-PET currently supports fat-tree and n-rooted-tree (including single and double rooted tree) topologies. Other topologies can be plugged in on demand.

All components in the data center model have certain restrictions. Links have a maximum bandwidth that restricts the amount of traffic that they can carry. Servers have a maximum processing capacity that restricts the work load that they can process.

3. Application Modeling

Applications in DC-PET are modeled as distributed collaborating components. Each component is called a *functional block*. Applications are represented as graphs of functional blocks. Analogous to the physical data center model, functional blocks (the graph's vertices) perform various functions and the links (the graph's edges) are used to exchange information between the functional blocks. Each link between two functional blocks represents a traffic "flow".

The links in application models are annotated with traffic figures. These figures describe how much traffic passes through the link in a particular direction as a response to a request coming to the application from the gateway. Each incoming request may result in a number of subsequent onward requests and ultimately in a response sent back to the originator of the initial request. For example, a Web server that receives a request for a particular Web page might need to fetch the page from a storage server and a number of data items to be included in that page from a data base server. Each incoming request therefore results in one onward request to a storage server and, for example, three requests to the data base server. The Web server then creates the Web page to be delivered to the user and sends back a response. All data exchanges can involve different amounts of traffic being exchanged, depending on the protocol used and the type of data being exchanged. The amount of traffic also linearly increases with the number of requests made to the application.

In real applications, many requests will not always occur, but might only happen with a certain probability, e.g. when data is cached. To accommodate this in applications models, in addition to the request size, there is the probability with which a particular request happens (averaged over a large number of requests). Both request size and request probability are part of application models. In Figure 1 they are attached to links as "R" and "S", meaning request probability and request size respectively.

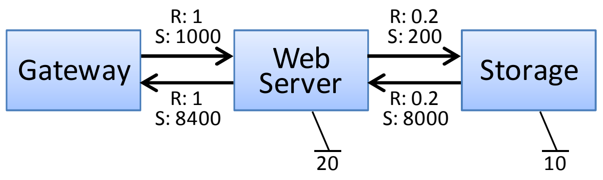

Figure 1: Example application model

Shown in Figure 1 is a Web server with a cache hit ratio of 80%. The remaining 20% of the requests need to be forwarded to storage (represented as a request probability of 0.2). As a different protocol is used for accessing storage, not only is the request probability lower, but so is the amount of data per request (200 bytes instead of 1000 for HTTP requests).

In addition to the traffic numbers on links, functional blocks are annotated with CPU figures. These figures specify how many requests can be handled by the functional block at maximum. The Web server component in Figure 1, for example, could handle 20 parallel requests, if deployed on its own server. The storage component, on the other hand, could handle only 10 parallel requests.

It is important to note that our model only describes statistical traffic and load averages, but not relationships between requests with respect to time.

4. Deployment

For the data center evaluation, applications are deployed onto the data center infrastructure. DC-PET maps the functional blocks of all applications to appropriate servers in the data center. Multiple functional blocks can be deployed on the same physical server, representing server virtualization.

The links between functional blocks are mapped to physical links. Each application layer link is mapped to one or more physical links and interconnecting devices. As data center topologies often offer multiple routes between servers, mapping to physical links has a certain freedom. The method that calculates and chooses particular routes is represented by the routing algorithm – both in the real world as well as in DC-PET. DC-PET allows the selection of particular routing algorithms for its evaluations. Algorithms might be selected depending on particular data center topologies or for fulfilling certain requirements. Through this mechanism, DC-PET allows the evaluation and comparison of different routing algorithms.

DC-PET can restrict assignment of functional blocks to certain servers, thereby modeling servers that have specific roles in the data center. The deployment of functional blocks can be restricted to certain server groups, making the servers of such groups assume particular roles. Several algorithms are available that cater for specific requirements on the selection of servers within a group (e.g., random, round-robin).

5. Design Options for Data Center Evaluation

Two different options – linear programming and packet-based simulation – have been identified for implementing the simulation functions. These two options, which both have their advantages and disadvantages, are briefly described in this section.

5.1. Option 1: Linear Programming

Linear Programming (LP) is a mathematical method for finding the optimum of a mathematical model (e.g., maximum number of requests handled in a data center). At its core, LP finds optimal values for a set of unknown variables under given constraints. As its name implies, LP requires the model to be formulated using only *linear* (in)equalities. Several LP solvers exist, commercial as well as open-source, which can solve LP problems with a very large number of unknowns in reasonable time.

LP can be used to model aggregated data center traffic and answer certain questions about this model. As an example, LP can be used to determine how many service requests a certain data center setup can handle.

5.2. Option 2: Packet-Based Network Simulation

For many years networks have been simulated by mimicking the packet transport between the nodes (e.g., server, switches) as well as the node behavior itself (delays, queues, network protocols, etc.). This approach allows for a very flexible, and – if necessary – very fine-grained and realistic simulation of all processes in the network. Furthermore, a variety of statistics can be collected during the simulation which allows the calculation of a large number of different networking and service metrics.

5.3. Assessment

Packet-based simulation is a proven technology that allows calculating a large variety of metrics. The main issue of a packet-based network simulation is that it is very time consuming. The time required for a simulation is, of course, dependent on the number of nodes and services to be simulated as well as on the level of detail used. If the level of detail is, however, reduced in order to shorten the simulation time, the accuracy of the simulation results might drop accordingly.

The main advantage of a data center evaluation tool based on linear programming is that it potentially supports analyzing several thousand servers and their services in a reasonable amount of time. All our simulations presented in section 9 run in under five minutes with most of them taking less than one minute.

The purely mathematical modeling of a data center with LP however has some limitations. These limitations mostly stem from the fact that for LP the formulation can use linear functions only. This limits the number of metrics which can be evaluated using LP. Certain metrics which might be interesting for a data center evaluation cannot be calculated by a linear mathematical formula (e.g., the number of dropped packets at a switch).

Another disadvantage of LP is that only a statistical abstraction of a real data center is modeled. While the results generated by the LP solver might be mathematically correct for the given LP model, such conditions might never exist in a real data center. Take, for example, a data center that hosts two identical applications. When finding the maximum number of requests that can be handled by the applications, one would expect each application to receive an equal share. In contrast, LP solvers will usually find a solution that apportions all requests to one application instance, leaving the other one with no requests. Although this is a mathematically correct solution, it might not occur in the real world.

For DC-PET, we chose to follow a linear programming based approach to estimating data center performance. The main reason was the much faster execution time of LP simulations when compared to a packet-based approach. It was decided that quick responses are more important than fine-grained results. To avoid extremely unfair traffic assignments, DC-PET includes some provisions that improve fairness of request and bandwidth assignments.

6. Simulation Process

An LP solver is a generic tool used to solve problems given in a specific mathematical form. The solver itself does not know about the physical data center, application models, and the traffic existing in the data center. These artifacts therefore need to be translated into a form that is suited for the LP solver.

The first step in our simulator is to merge the application models and the physical data center model. As shown in Figure 2, this is done by deploying the applications on the data center meaning that the functional blocks of applications are assigned to servers. Next, flows of traffic between the functional application blocks are created. As these flows might traverse multiple physical devices, the flows must be routed appropriately.

Routing is the only network-related function that is still explicitly modeled in the LP-based data center evaluation tool. Although the routing algorithm is not applied to each and every packet (as there are no individual packets in the LP problem), the assignment of traffic flows (i.e., chains of packets) to physical links through the network is still handled by the routing algorithm. As this assignment only happens once when setting up the LP problem, it does not exhibit any dynamic properties and therefore assumes a static network that does not exhibit any link or device failures. Although this is not the case in reality, it is a reasonable assumption for approximating traffic load on links. As the routing algorithm affects the assignment of traffic to links, it has a large influence on the traffic load and therefore on the performance of the data center.

The results of deployment and routing are expressed as linear equations which are one main input to the LP solver.

The other main input to the LP solver are the constraints on link bandwidth and server CPU capacity. It must be guaranteed that the combined traffic on a physical link does not exceed the capacity of that link. Furthermore, the capacity of the servers should not be exceeded by the applications assigned to them. These restrictions can be modeled as specific inequalities.

DC-PET automatically creates the equations needed for expressing deployment, traffic routes, and capacity restrictions. The created LP formulation can then be worked on by an LP solver which takes care of finding the actual solution for the specified problem.

Once a solution has been found, the LP solver's results are transformed back into the data center and application model space. Additionally, metrics which have not been directly calculated by the LP solver are derived from its results, e.g. link utilization. These two steps are performed in the "Post-processing" step.

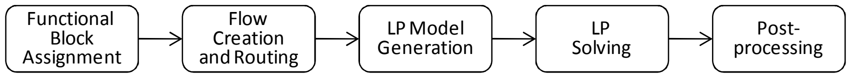

Figure 2: Simulation Process

7. Linear Programming Formulation

In this section, we provide the LP formulation used by DC-PET. DC-PET transforms the given network and application models into a corresponding LP formulation and – after running the LP solver – extracts the results and translates them back into the model domain.

The *objective function* of the LP system is to maximize the total number of requests which are handled by the data center:

$$\text{maximize} \sum_{a \in \mathbf{A}} R_a$$

with **A** being the set of all applications deployed and R_a being the number of requests handled for application a, $a \in \mathbf{A}$. This maximization is subject to several conditions.

The first set of conditions ensures that no link in the network is overloaded (link capacity bounds)

$$\sum_{i \in \mathbf{U}_l} F_i \leq c_l \qquad \forall l \in \mathbf{L}$$

with F_i being the traffic rate for flow i, **L** being the set of all links in the networks, $\mathbf{U}_l$ being the set of all flows traversing link l, and c_l being the capacity of link l.

Furthermore, the flow rates F_i are linked to the application requests R_a rates via linear weights $w_{i,a}$:

$$F_i = w_{i,a} R_a \qquad a \in \mathbf{A},\ i \in \mathbf{\Phi}_a ,$$

with $\mathbf{\Phi}_a$ being the set of flows belonging to application a.

Finally, the servers' CPU utilization must not be larger than the servers' processing capacity, which leads to the following set of conditions:

$$\sum_{a \in \mathbf{A}} u_{a,n} R_a \leq x_n \qquad \forall n \in \mathbf{N},$$

with **N** being the set of all server nodes, x_n the processing capacity of node n, and $u_{a,n}$ being the CPU load caused on node n for each processed request of application a.

From the above equations it can be seen that the LP solver needs to maximize the objective function only

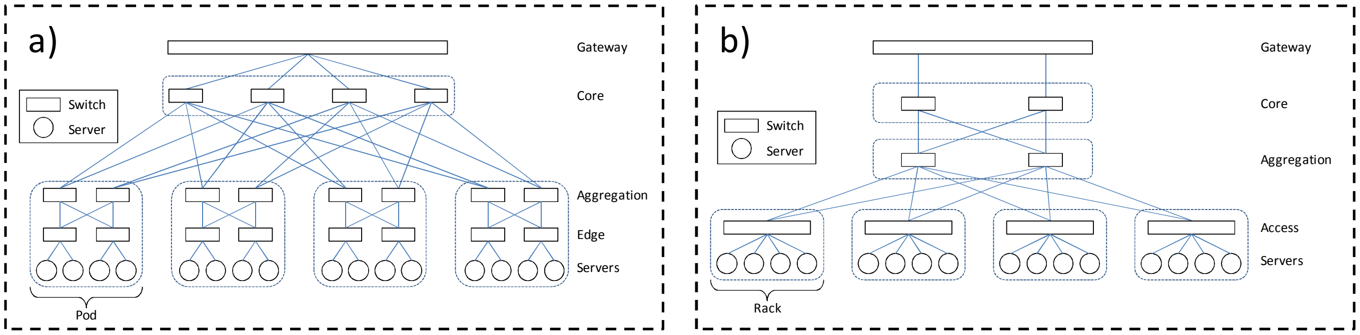

Figure 3: Data Center Topologies: a) Fat Tree; b) Double-Rooted Tree

with respect to the application request rates R_a. Values for $u_{a,n}$ are calculated from CPU utilization figures in application models and their assignment to servers. Values for $w_{i,a}$ are calculated from bandwidth consumption figures in application models. The set U_l is derived from the routing of the flows between servers the functional blocks are deployed on, and x_n are constants which can be taken directly from the network model.

8. Metrics

After the LP evaluation, its results (application request rates and flow rates) are transformed into a number of metrics that are potentially relevant for assessing the performance of a data center. Metrics pertaining to links and devices (switches, servers) can be distinguished.

8.1. Link Metrics

Data rate: This metric represents the amount of data that is transported over a link. This measurement is attached to a unidirectional link, specifying the traffic that crosses that physical link in a particular direction. This measurement is the same as the traffic-in and traffic-out values on the ports connected to the endpoints of this link.

Number of flows: This metric represents the number of parallel connections that are crossing a link in one direction. Each flow relates to one connection in one direction between two functional application blocks.

8.2. Device Metrics

Traffic-in(-out) rate: This metric represents the amount of data that is arriving at (emitted from) a particular data center element. The metric is associated with a port, which may belong to a switch, a server, or a gateway. Elements that contain one or more ports possess aggregate measurements of the traffic-in (out) rate.

Flows in (out): This metric represents the number of parallel connections that are receiving (sending) traffic from (to) a particular port. Each connection relates to one connection in one direction between two functional blocks.

CPU utilization: This metric represents the utilization of the CPU of a particular server (between 0% and 100%).

9. Experiments

The previous sections introduced the modeling abstractions that DC-PET uses to describe data centers and applications. This chapter details the DC topology and application models that were evaluated with DC-PET and shows some results.

9.1. Data Center Topologies

Our simulations were run on two different networking topologies: a fat tree and a double-rooted tree.

Fat-tree networks are oversubscription-free topologies, meaning that they have the same aggregate bandwidth between all layers. They can be built in the form of Clos networks from switches that all have the same number of ports and link bandwidth [1]. The main parameter for the size of the network, in particular for the number of supported servers, is the number of ports of each switch. An example fat-tree topology constructed from 4-port switches is shown in Figure 3a. The topology is structured into five layers: the server, the edge, the aggregation, the core, and the gateway layer. In addition to the traditional four layers of the fat tree, we introduced the gateway layer to model connections to the outside world. This layer is somewhat different from the others, in the sense that the switch used there can have a different number of ports and the links connecting to it can have different bandwidths than the other connections in the data center.

Routing in a fat tree is done by moving up and down the tree depending on the locations of the traffic source and destination. There exist multiple paths between each layer of the tree. The simplest way to distribute traffic across these paths is to choose a random link at each layer whenever multiple links exist which lead to the destination without increasing the path length.

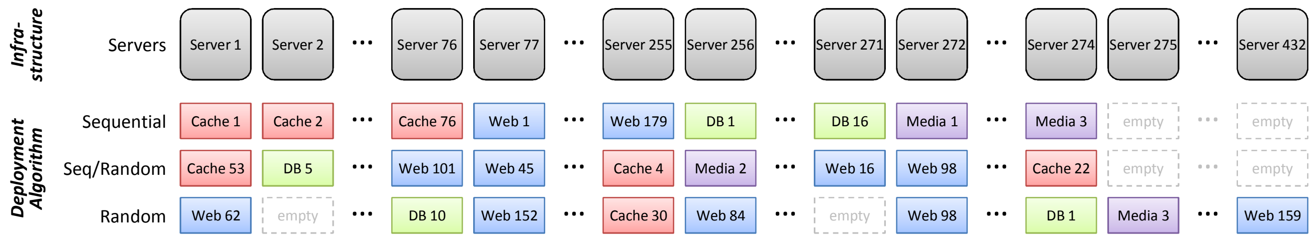

Figure 4: Deployment scenarios evaluated

Double-rooted trees are the currently prevailing topology for data center networks. As shown in Figure 3b, they consist of a number of layers, namely the access, aggregation, and core layers [2]. Servers are part of the access layer and are connected to a top-of-rack switch, usually with a single uplink. From there, the two roots emerge, with both being able to replace each other for redundancy. The aggregation layer concentrates the traffic from the access switches and passes it on to the core network of the data center.

To cater for the concentration of traffic towards the root, the links between the different layers in the double-rooted-tree topology usually provide increasing bandwidth capacities towards the root.

9.2. Application Model: Wikipedia

The example application model that we used for evaluating data center performance is a Web serving example modeled closely after Wikipedia (see Figure 5). It resembles the structure, traffic patterns, and sizes from Wikipedia as closely as possible. The model is simplified from the real Wikipedia architecture, mostly be removing the functional blocks that do not deal with the (arguably) main task of Wikipedia of serving Web pages. The links and functional blocks are annotated with statistical numbers taken from various sources on the Web[1].

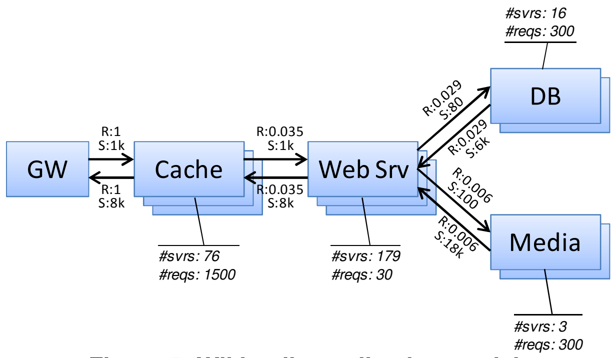

Figure 5: Wikipedia application model

[1] The Wikipedia architecture is shown in http://upload.wikimedia.org/wikipedia/commons/f/ff/Wikimedia-servers-2008-11-10.svg, and a number of statistics are given at http://old.nabble.com/wikipedia-servers-and-statistics-td21186836.html.

Most functional blocks are replicated within the Wikipedia model. Such replication is usually introduced for load balancing reasons, in order to increase the capacity of the function provided by that block. Traffic going into such functional blocks is distributed among the participating instances according to some scheme. Various schemes are possible. For our experiments, we chose a uniform distribution, i.e., every functional block of a certain type processes exactly the same amount of traffic.

9.3. Setup

We evaluated the Wikipedia application as described in the previous section on fat-tree and double-rooted-tree networks. All links in the fat tree have a capacity of 1 Gbps while the links in the double-rooted tree have capacities of 1 Gbps between the servers and the top-of-rack (ToR) switches and 10 Gbps between the ToR and the aggregation switches. The links to the gateway layer have unlimited capacity in both topologies.

The fat-tree topology has been evaluated in two sizes. One setup is constructed from 12-port switches (FT12), the other one from 24-port switches (FT24). The data centers host 432 and 3456 servers, respectively. The double-rooted tree (DRT) consists of nine racks with 48 servers each (and a single 48-port ToR switch per rack), giving the setup the same number of servers (432) as the small fat-tree scenario.

In the experiments, every server hosts one functional block at the maximum. We used three different deployment algorithms: *sequential*, *sequential/random*, and *random*. The *sequential* deployment algorithm selects servers for functional blocks in a purely sequential fashion, meaning that all functional blocks of a particular type are deployed "next" to each other (see Figure 4). Mapped to the data center topology, this means that some racks/pods are filled completely with one type of functional blocks (FBs), namely caches and web servers. Uplinks from these racks/pods therefore carry traffic from these FBs only.

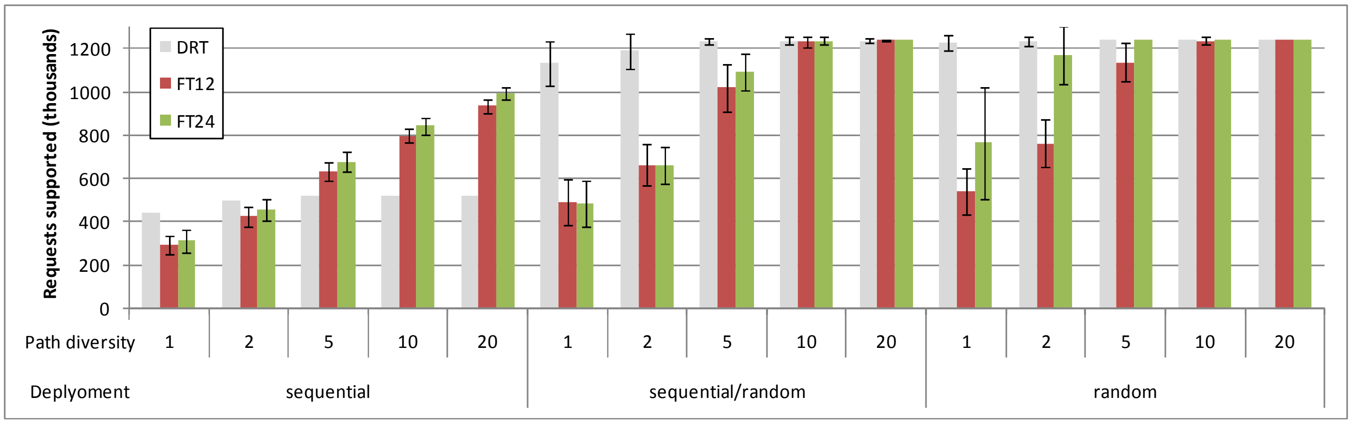

Figure 6: Average number of requests supported

The *sequential/random* deployment still sequentially selects servers (meaning that computation and traffic load are aggregated in a certain part of the data center), but assigns functional blocks in a random fashion. Different traffic originating from the different types of FBs is therefore spread more equally across the data center, potentially resulting in better performance.

The *random* deployment selects servers for functional blocks randomly from the complete set of servers. As in our experiments only 274 FBs need to be assigned while the topologies provide 432 or 3456 servers, the computing and network load is spread more widely.

In addition, we varied the path diversity between 1 and 20, meaning that the traffic carried by a link between two functional blocks is distributed across 1 to 20 (randomly chosen) routes between data center components. Path diversity decreases the chances for overloading single links and should therefore ensure higher throughput and with it higher performance of the data center.

9.4. Results

Our experiments provided a number of interesting results. First, running the simulations with bounded CPU capacity leads to the same performance numbers for all setups. There is no influence of the selected network topology or the path diversity. The CPU is the bottleneck and restricts the maximum throughput.

Next, we changed the CPU capacity to infinity in order to find bottlenecks in the network. The limiting factor now are the links between caches and the gateway. These links are saturated at an incoming request rate of about 1.2M (distributed among 76 cache instances). Looking at the results shown in Figure 6, it can be seen that this maximum is not always reached. Note that all simulations were run 50 times with different random number sequences and then averaged. The standard deviations are shown in the figure as black bars on top of the measurements.

From the figure it can be observed that the deployment algorithm plays an important role in the attainable performance. Spreading functional blocks and thereby distributing network load more equally increases overall throughput. The second observation is that adding diversity to flows increases throughput. This effect is more pronounced in fat-tree topologies than in the double-rooted tree. The third observation is that the double-rooted tree performs worse with sequential deployment due to fully utilized links between the access and the aggregation layers because of the oversubscription ratio. Another interesting observation is the high standard deviation for the FT24 low diversity deployments. It highlights that some random placements perform much worse than others. Path diversity is therefore a necessity for efficiently utilizing fat-tree networks.

Digging deeper into the simulation results (not shown here) reveals that in the double-rooted tree, links between access and aggregation layer are utilized most (average: about 60%), compared to about 10% average utilization at the access/server level. Looking at the distribution of link utilization within the access level reveals an uneven distribution: some links are fully utilized while others are not used at all. Utilization of links in a 12-ary fat tree is more evenly distributed with the average maximum close to 10% on all layers.

10. Related Work

Linear Programming (LP) is a commonly used technique for solving optimization problems in various domains. It has been used for optimizing electrical power networks [3] and has also been applied to communication networks, especially in the areas of QoS routing [4][5] and joint scheduling and routing problems [6]. The LP formulations often make use of integer variables leading to an Integer Linear Programming or Mixed Integer Linear Programming problem. Both types are known to be NP hard.

Instead of using LP, simulations of the behavior of communication networks often use event-based simulations on the level of individual packets, including [8]. Most common is the use of ns-2, an event-based simulator. Packet-level simulations are trying to reproduce network behavior as closely as possible and can therefore produce detailed and accurate results. Besides their disadvantage of requiring large amounts of time to generate results, packet-level simulators cannot determine the maximum supported load of given configurations as can be done with the linear programming approach of DC-PET.

The majority of simulations are restricted to assessing the performance of the networking domain only. Some work on simulating job scheduling also exists. Although Buyya et al. claim to simulate both network and servers, the results presented in [7] relate to the server domain only. Kliazovich et al. simulate both networking and server domain, but their focus is on energy consumption [8]. DC-PET is based on the principle of co-simulating both network and server domains for performance assessment.

To the best of our knowledge we are the first to develop an LP-based network and service simulator for large-scale data centers.

11. Conclusion

Designing data centers with the right size is hard. Even if incoming traffic is known or at least appropriately estimated, the traffic arising inside the data center between application components and the interactions of traffic within and across applications are hardly known and make sizing of applications and data centers difficult. We developed a data center planning and evaluation tool called “DC-PET” which aims at helping designing a data center by assessing the performance of design alternatives.

DC-PET uses linear programming to perform its data center analysis. LP has the main advantage of quickly providing results. On the downside, the analysis happens on a statistically abstracted level and does therefore provide only limited detail. Our experience is that the results are helpful for sizing applications and for comparing different deployment options.

So far, we only simulated the load of a large-scale Web application. In the future, we plan to add other applications to the portfolio and simulate their behavior both in isolation as well as in combination with other applications. The results are expected to improve the quality provided by data centers while minimizing costs helping DC-PET’s users increase their competitiveness.

12. References

[1] Al-Fares, M., Loukissas, A., and Vahdat, A., "A scalable, commodity data center network architecture", *SIGCOMM Comput. Commun. Rev.*, 38, 4, 2008, pp. 63-74.

[2] Cisco Systems Inc., *Cisco Data Center Infrastructure 2.5 Design Guide*, 2007. http://www.cisco.com/application/pdf/en/us/guest/netsol/ns107/c649/ccmigration_09186a008073377d.pdf

[3] Garver, L. L., “Transmission Network Estimation Using Linear Programming”, *IEEE Transactions on Power Apparatus and Systems*, PAS-89, 7, September 1970, pp. 1688-1697.

[4] Xiao, Y, Thulasiraman, K., and Xue, G., “QoS Routing in Communication Networks: Approximation Algorithms Based on the Primal Simplex Method of Linear Programming”, *IEEE Transactions on Computers,* 55, 7, July 2006, pp. 815-829.

[5] Chang, J., Tassiulas, L., “Maximum lifetime routing in wireless sensor networks”, *IEEE Transactions on Networking*, 12, 4, August 2004, pp. 609-619.

[6] Wang, Y., Wang, W., Li, X., Song, W., “Interference-Aware Joint Routing and TDMA Link Scheduling for Static Wireless Networks”, *IEEE Transactions on Parallel and Distributed Systems*, 16, 12, December 2008, pp. 1709-1725.

[7] Buyya, R., Ranjan. R., and Calheiros, R. N., Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities, in *Proceedings International Conference on High Performance Computing & Simulation*, Leipzig, Germany, June 21-24 2009.

[8] Kliazovich, D., Bouvry, P., Audzevich, Y., and Khan, S. U., “GreenCloud: A Packet-level Simulator of Energy-aware Cloud Computing Data Centers”, in *Proceedings IEEE Globecom 2010*, Miami, FL, USA, December 6-10 2010.