Proactive Incast Congestion Control in A Datacenter Serving Web Applications

Haoyu Wang, Haiying Shen
University of Virginia
hw8c, hs6ms@virginia.edu

Yuhua Lin
Clemson University
yuhual@clemson.edu

Background and Motivation

A data query for a web application always needs to retrieve many data objects concurrently from different cloud servers which may cause incast congestion, a non-ignorable reason of the delay in datacenter.

The root cause of the incast congestion problem is the many-to-one concurrent communications between the single front-end server and multiple data servers.

In PICC, the front-end server gathers popular data objects (i.e., frequently requested data objects) into as few data servers as possible. It also re-allocates the data objects that are likely to be concurrently or sequentially requested (called correlated data objects) into the same server.

Current solutions

Previous incast problem solutions usually consider the data transmission between the data servers and the front-end server directly. They cannot proactively avoid the situation that a large number of data servers are concurrently connected to the front-end server only a few data objects from each data server. There are many works focusing on the incast congestion control problem, which can be classified into three groups: link layer solutions, transport layer solutions and application layer solutions.

Our approach

PICC: Proactive Incast Congestion Control System

In PICC, the front-end server gathers popular data objects into as few data servers as possible. It also re-allocates the data objects that are likely to be concurrently or sequentially requested (called correlated data objects) into the same server. As a result, PICC reduces the number of data servers concurrently connected to the front-end server, and the number of establishments of the connections between data servers and the front-end server, which avoids the incast congestion and reduces the network latency.

Design Details

CSA novelly changes data-allocation-first to **job-scheduling-first**. Job-scheduling-first enables CSA to proactively consolidates tasks with more common requested data to the same server when conducting deadline-aware scheduling, and also consolidate the tasks to as few servers as possible to maximize energy savings.

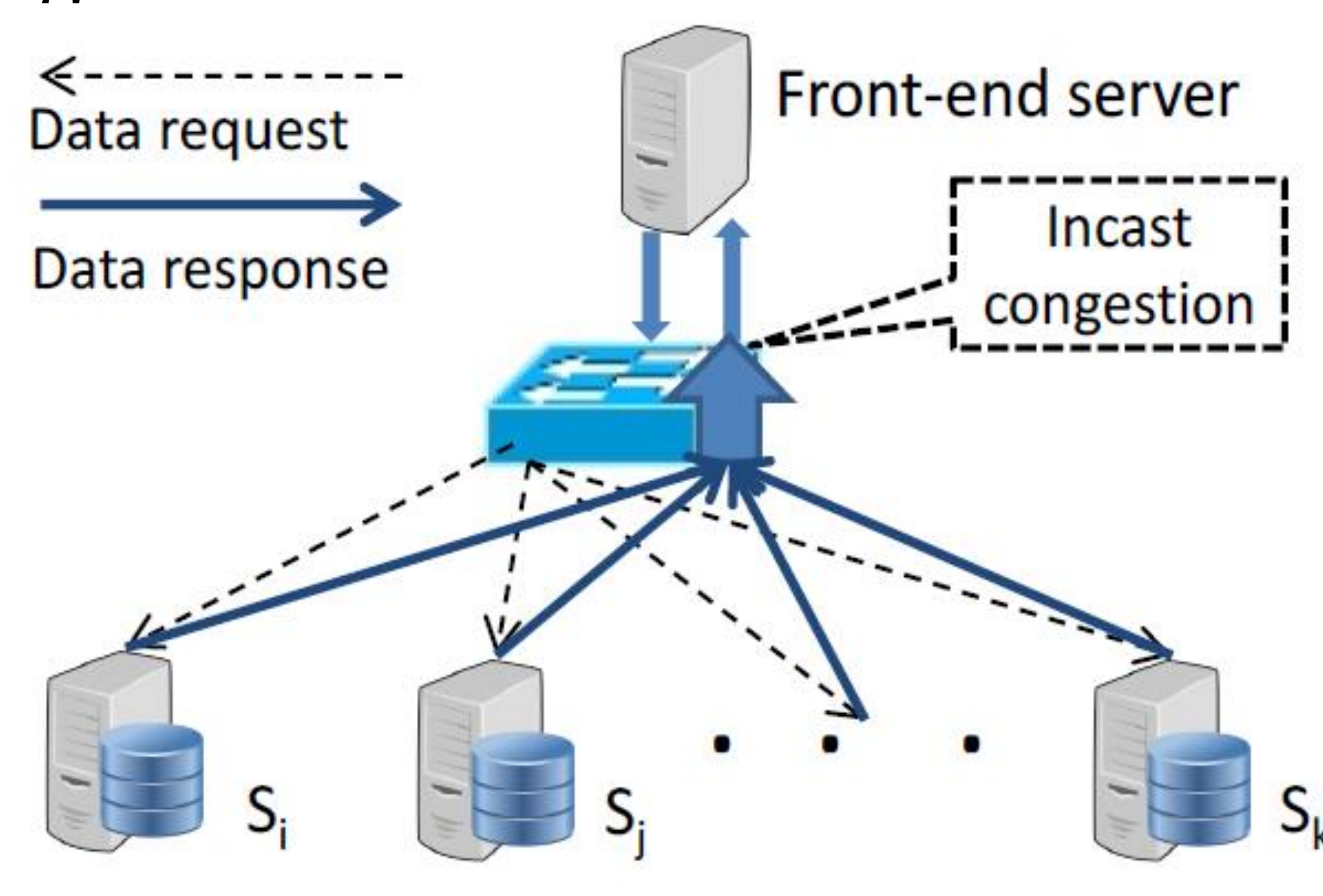


Figure 1: Illustration of datacenter incast congestion.

Scheme 1: Popular Data Object Gathering

- The popular data object gathering method gathers popular data objects into as few data servers as possible. As a result, there is a higher probability that only a limited number of data servers respond to the front-end server, so that the incast congestion has a lower probability to occur.

Scheme 2: Correlated Data Object Gathering

- We propose the correlated data object gathering method to cluster the concurrently or sequentially requested data objects (i.e., correlated data objects) into the same group and allocate the group of correlated data objects into the same gathering server.

Scheme 3: Queuing Delay Reduction

- The gathering Server above maintains a sending queue of all requested data objects and sends them out sequentially. In order to minimize the average waiting and transmission time per data object, we can reduce the effect of head-of-line blocking by setting different priorities for the data objects based on their sizes and waiting times.

Experimental Results

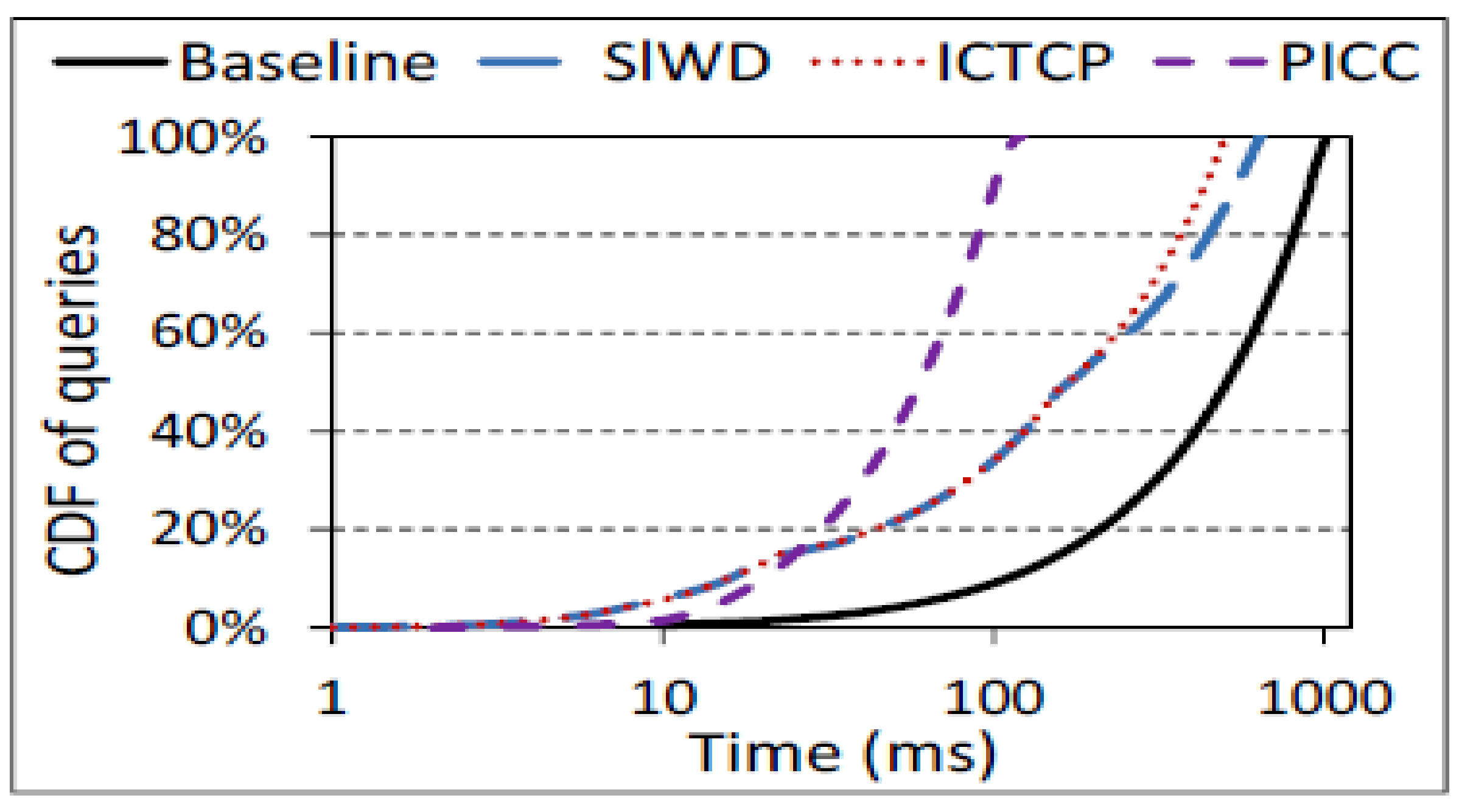


Figure 2 CDF of query latency.

Result: Lowest query latency.

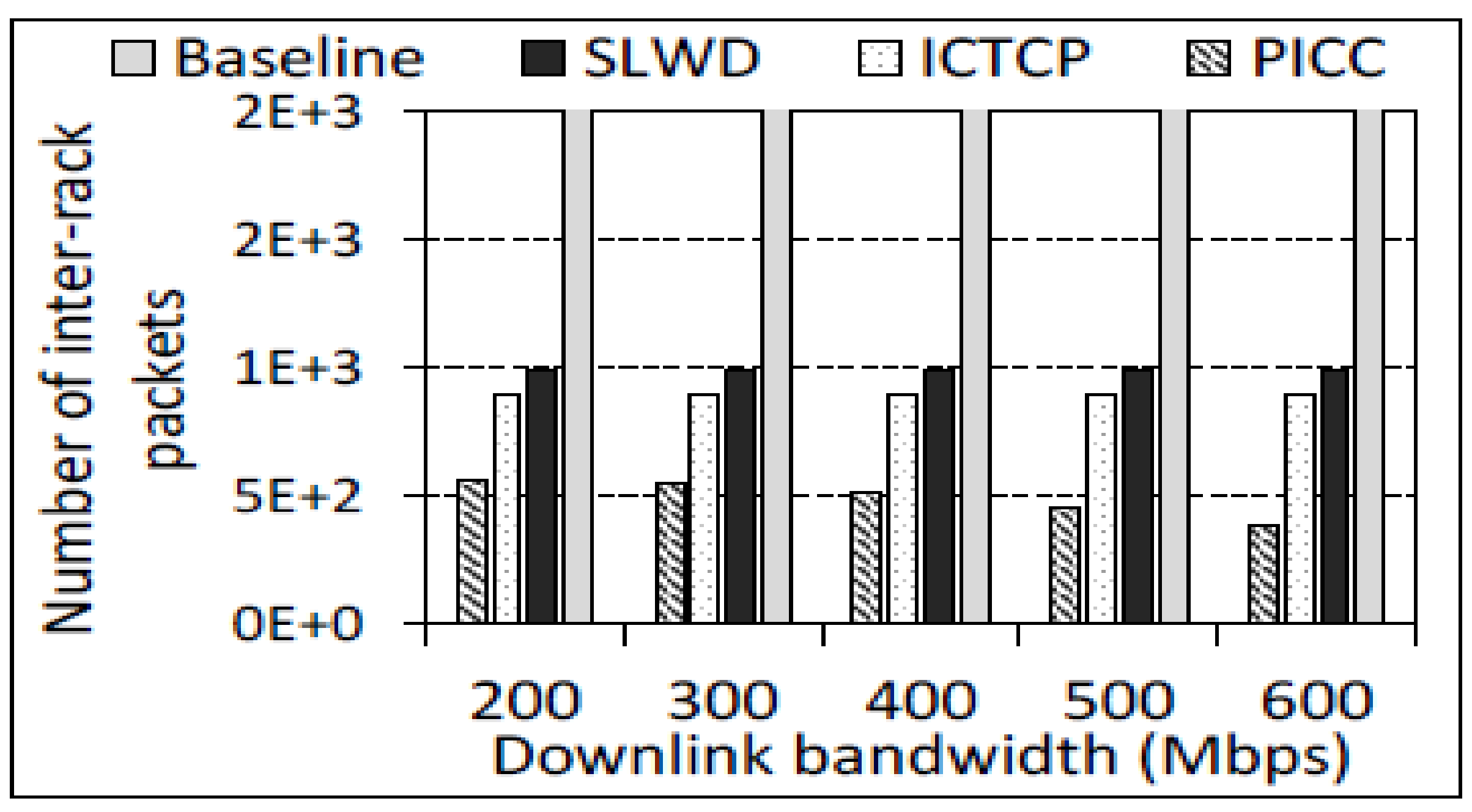
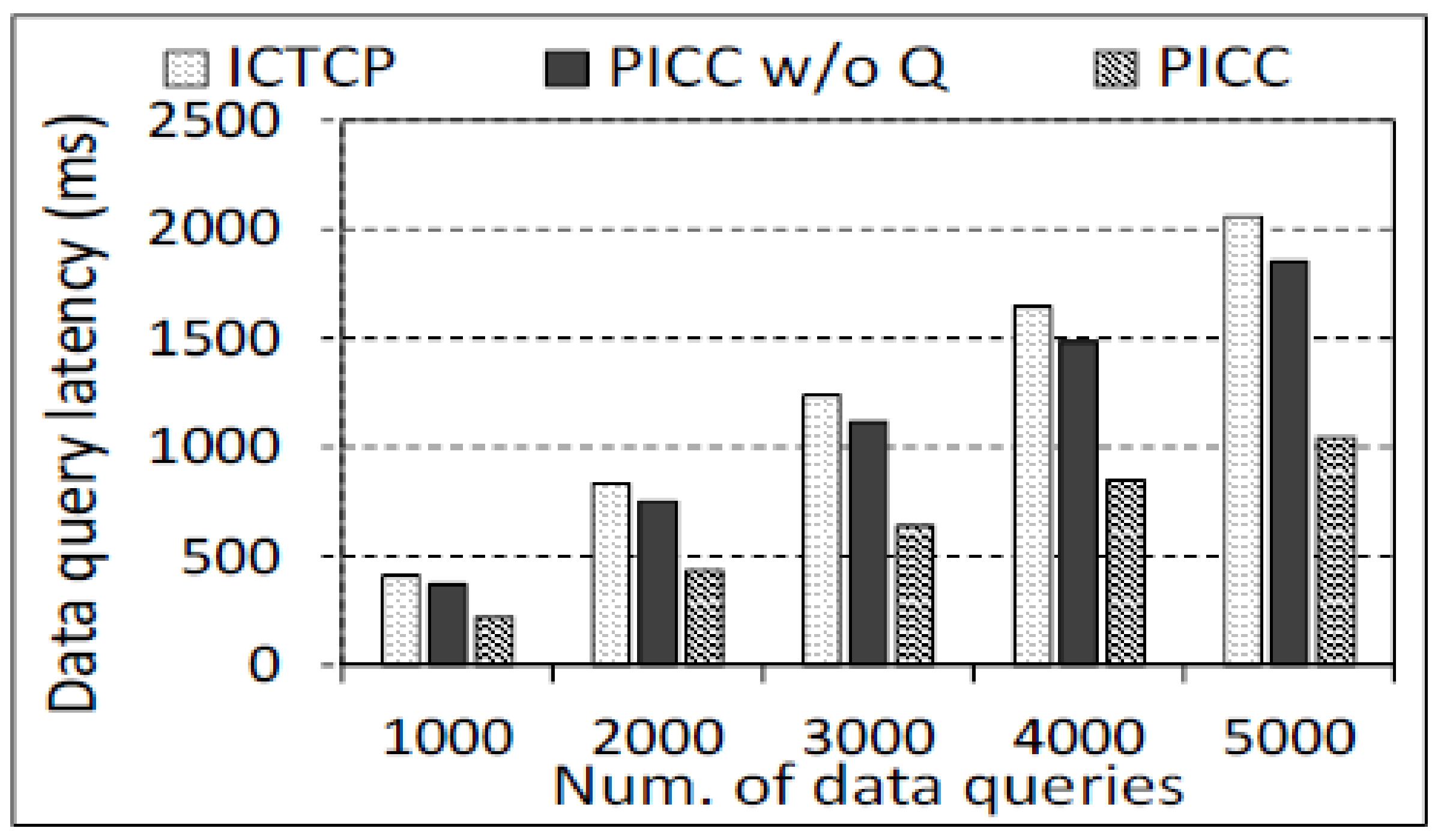


Figure 3 Inter-rack data transmission.

Result: Best data locality performance.



Result: Effectiveness of queuing delay reduction method .

References:
[1] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani. Scaling Memcache at Facebook. In Proc. of NSDI, 2013.

Future Work

In the future, we will consider how to reallocate data objects more efficiently and how to integrate different application level solutions in order to further reduce the data query latency.

Acknowledgments

U.S. NSF grants NSF-1404981, IIS-1354123, CNS-1254006, IBM Faculty Award 5501145 and Microsoft Research Faculty Fellowship 8300751.