Solution Brief

Intel Infrastructure Processing Unit

Intel Infrastructure Processing Units (IPUs) Leverage Napatech's Virtualized Data Plane Software to Enable Breakthrough Performance for Microservices-based Cloud Applications

napa:tech;
SMARTNIC SOLUTIONS

Cloud service providers, including well-known companies such as Amazon, eBay, Netflix, and Twitter, are increasingly implementing their applications as microservices rather than traditional monolithic designs. While microservices-based software architectures deliver important benefits such as accelerated deployment, simplified debugging and improved scalability, they introduce significant networking overheads so that parameters such as network latency have a major influence on overall application performance and data center cost. By leveraging Intel® FPGA Infrastructure Processing Units (IPUs) running virtualized data plane software from Napatech, service providers can maximize the performance of their networking infrastructure, enabling a level of performance otherwise unachievable while minimizing their overall data center CAPEX and OPEX.

This solution brief analyzes the benefits delivered by an IPU solution for two microservices-based use cases, a Publish-Subscribe (Pub-Sub) application and a three-tier client-server application. In these use cases, the IPU solution enables a 50% increase in system throughput compared to a system configuration based on a standard Network Interface Card (NIC).

Microservices: A Modern Software Architecture

Rather than implementing applications as traditional monolithic software systems, cloud service providers are increasingly adopting a "microservices" architecture. This modern software development approach involves breaking down a large application into smaller, independent, loosely coupled services. Some of the benefits of microservices include:

- **Scalability:** Microservices allow for better scalability, as individual services can be scaled independently of each other. This means that developers can scale up or down only the services that need it, without affecting the rest of the application.
- **Flexibility:** Microservices make it easier to make changes to a system because individual services can be updated without affecting the entire application. This makes it easier to adopt new technologies, experiment with different programming languages and test new features.
- **Fault isolation:** Since each microservice is an independent component, if one service fails it doesn't affect the rest of the application. This means that developers can quickly identify and fix problems without affecting the entire system.
- **Improved development speed:** Microservices enable smaller, more focused development teams to work independently on specific services. This speeds up the development process and makes it easier to manage large, complex systems.
- **Better fault tolerance:** With microservices, it's easier to build fault-tolerant systems because each service can be designed to handle errors independently. This means that the entire system is more resilient and less likely to fail.
- **Improved testing:** Since each microservice is independent, it's easier to test individual services. This means that developers can test services in isolation, which makes it easier to find and fix bugs.

Authors

Charlie Ashton
Senior Director of Business Development
Napatech

Rich Howell
Product Marketing Manager
Intel Corporation

As one example, available reports (e.g. "System Design Netflix – A Complete Architecture") explain that Netflix adopted Amazon Web Services (AWS) for managing their IT infrastructure, replacing their existing monolithic programs hosted on their own data servers with a microservices architecture hosted in the public cloud. This enabled them to deploy an extremely scalable IT infrastructure with support for millions of service requests.

In the microservices-based architecture deployed by Netflix, larger software programs are broken down into smaller programs, or components, based on modularity. Every such component has its own data encapsulation. Netflix is able to scale its services rapidly, via horizontal scaling and workload partitioning as part of the microservices-based architecture. If any smaller software program stops working or starts slowing down system requests, engineers can quickly isolate that component and ensure uninterrupted service. The microservices-based architecture also enables tracking of every individual software component.

Networking Challenges for Microservices

In a microservices architecture, however, network latency presents a significant challenge as virtualized services implemented in containers or Virtual Machines (VMs) communicate with each other over a virtualized network. For example, microservices communicate with each other frequently, which can result in a large amount of network traffic. This increased network traffic can lead to network congestion and increased latency, which can negatively impact the performance of the system. Similarly, in a microservices architecture, services often need to call other services to complete a task and each network call adds additional latency to the system. As the number of services and the complexity of the system increases, the number of network calls also increases, which can lead to significant latency challenges. Finally, different microservices may use different network protocols for communication. For example, one service may use REST (REpresentational State Transfer) while another service may use gRPC (Google Remote Procedure Call). Translating between different network protocols can add additional latency to the system.

Traditionally, a virtualized data plane is implemented completely in software and many of its compute cycles are consumed by running a virtual switch (vSwitch) which routes network traffic between VMs. Since each vSwitch operation requires a significant number of CPU cycles, this architecture can introduce unacceptable latency into the system and may also prevent the system from achieving the overall performance or throughput required. At the same time, a CPU that is heavily utilized running the virtual data plane will have fewer cores available for running applications and services, increasing the number of servers required to support the data center workload, and increasing both CAPEX and OPEX.

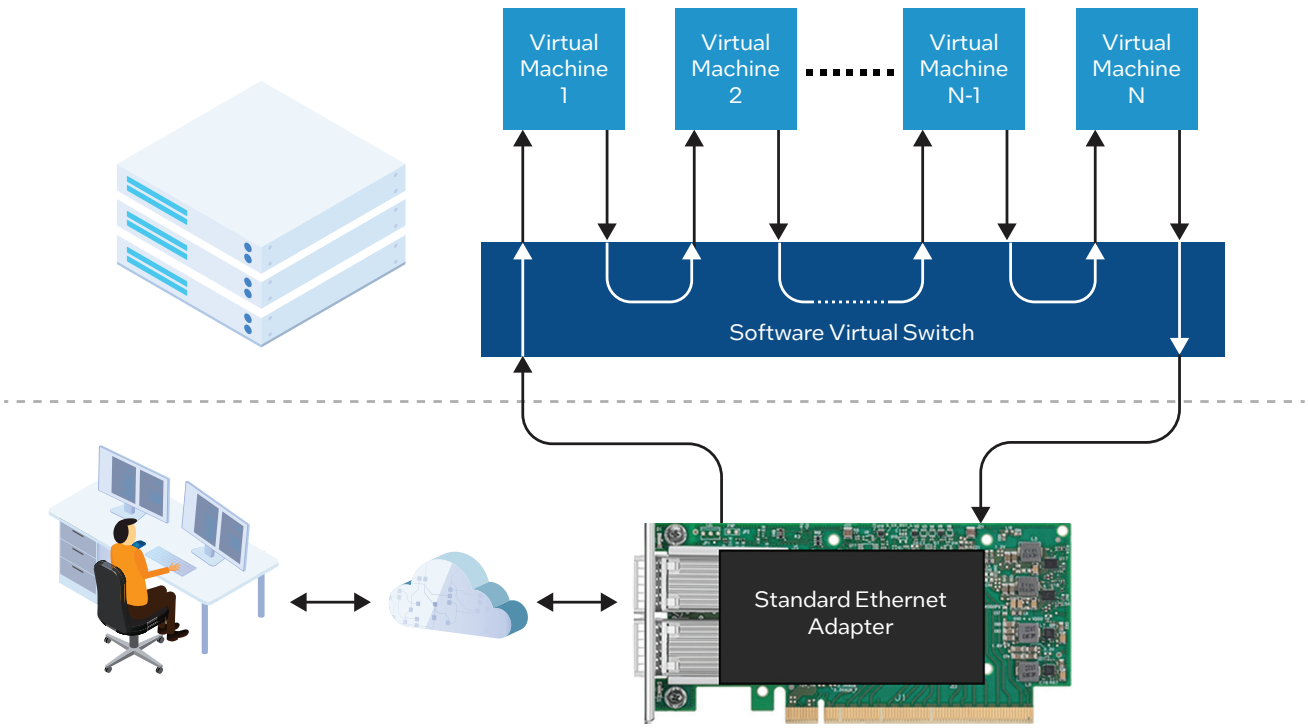


Figure 1. Multiple VMs interconnected with a software virtual switch running on a server configured with a standard Ethernet adapter

The Advantages of an IPU

A more efficient and cost-effective system-level architecture leverages an Intel FPGA IPU to offload the vSwitch from the server CPU, freeing up the server CPU for running applications and services.

The IPU, which replaces the standard Network Interface Card (NIC) in the data center server, implements the vSwitch in hardware, using a programmable FPGA (Field-Programmable Gate Array) to run the data plane in conjunction with a general-purpose CPU that runs the control plane. The vSwitch presents an industry-standard application programming interface (API) to the VMs, ensuring that no changes need to be made to the VMs themselves when taking advantage of this architecture.

The IPU-based architecture delivers three key benefits for a data center running microservices-based applications:

- Ultra-low latency, which minimizes the delayed traffic between the microservices;
- High performance, which maximizes the overall throughput of the system and application;
- Optimum server CPU utilization with no server CPU cores consumed by the vSwitch data plane, which minimizes the total number of servers required for the overall workload, also minimizing data center CAPEX and OPEX.

MIT Analysis

To quantify the benefits of vSwitch offload in real-world scenarios, Massachusetts Institute of Technology (MIT) analyzed the performance of two microservices-based use cases, comparing the results from using a traditional software-based vSwitch with those obtained using an Intel IPU running virtualized data plane software from Napatech, a leading provider of SmartNIC and IPU solutions. These two use cases were a publish-subscribe "pub-sub" application that uses message passing for data transfers across multiple tiers and a three-tier TCP application comprising a web server, in-memory cache, and back-end database.

The results of this benchmarking initiative are documented in the paper "Microservice Benchmarking on Intel IPUs running Napatech Software" published by MIT.

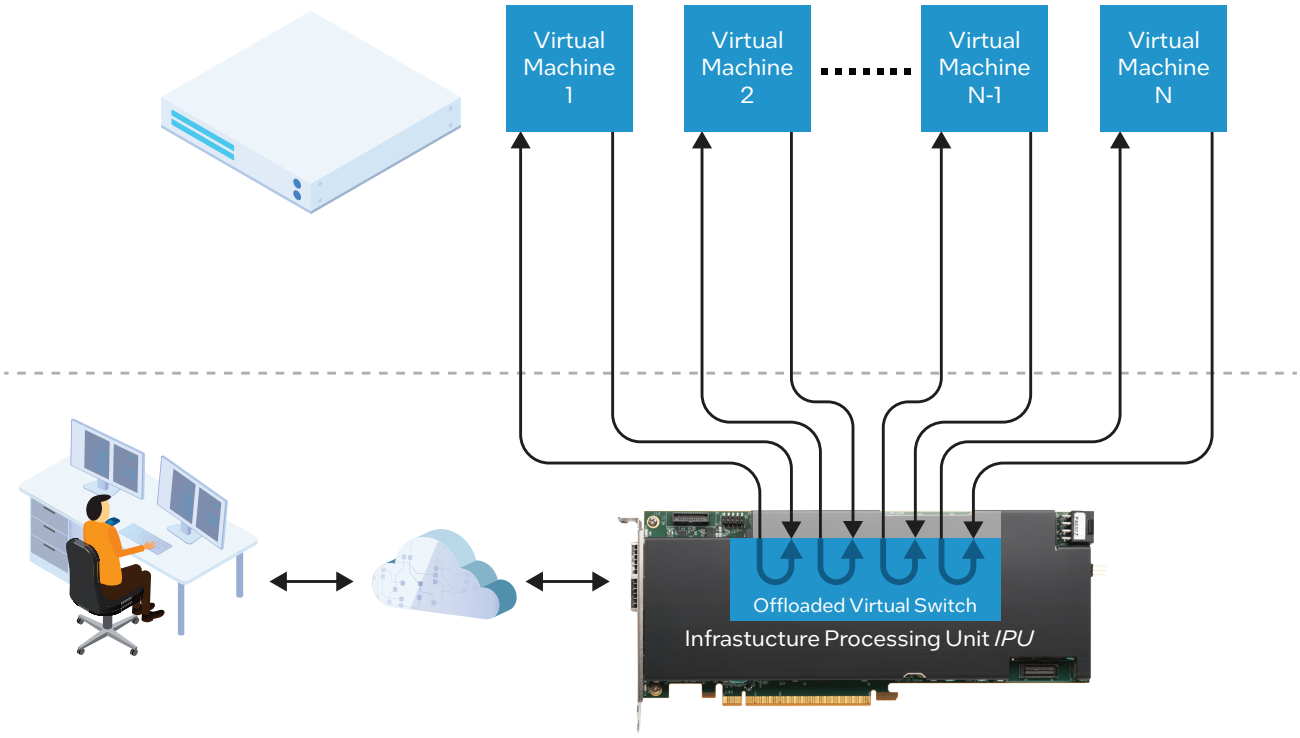


Figure 2. High-performance system architecture in which multiple VMs are interconnected with an offloaded virtual switch running on an IPU

Pub-sub Application Performance

A pub-sub application, short for "publish-subscribe application," is a messaging pattern commonly used in distributed systems to facilitate communication and coordination between different components or services. The pub-sub pattern allows for asynchronous and decoupled communication, where senders of messages, known as publishers, do not need to know the specific recipients, known as subscribers. Pub-sub applications are applicable to use cases such as:

- **Seating reservation systems** that create a floor plan, assign seats to it, and then manage the live seat-booking events. As clients buy tickets, the pub-sub system updates the floor plan everywhere in real time and keeps the distributed cache system in sync. Clients never end up requesting a seat only to find out someone had bought it while they were still in the browsing/shopping phase.
- **Educational tools** that allow students to participate in a classroom via a web-based app, where clients often encounter issues such as unreliable WiFi or unpredictable cellular networks. The pub-sub system recovers its connection when they rejoin the network and is able to handle rapid changes in the number of online participants.
- **Financial applications** such as the distribution of market data including stock prices, market indices, trade data, and order book updates to subscribers within an organization.
- **Internet of Things (IoT) systems**, where pub-sub facilitates communication between numerous IoT devices and enables efficient data dissemination. Sensors publish data, then subscribers can receive and process that data in real-time.

For this analysis, MIT evaluated a five-tier chain topology developed with a pub-sub communication model from Dapr, which is a portable, event-driven runtime that enables developers to build resilient, stateless and stateful applications that run both on the cloud and edge, while supporting a diversity of languages and developer frameworks. Each tier performs CPU-intensive computation for a user-specified amount of time, before broadcasting its output to the downstream tier.

Within the five-tier pub-sub application, the placement of services across the two OVS-enabled servers ensures that dependent services are running on different physical machines, so that all traffic between tiers passes across the IPUs, when enabled.

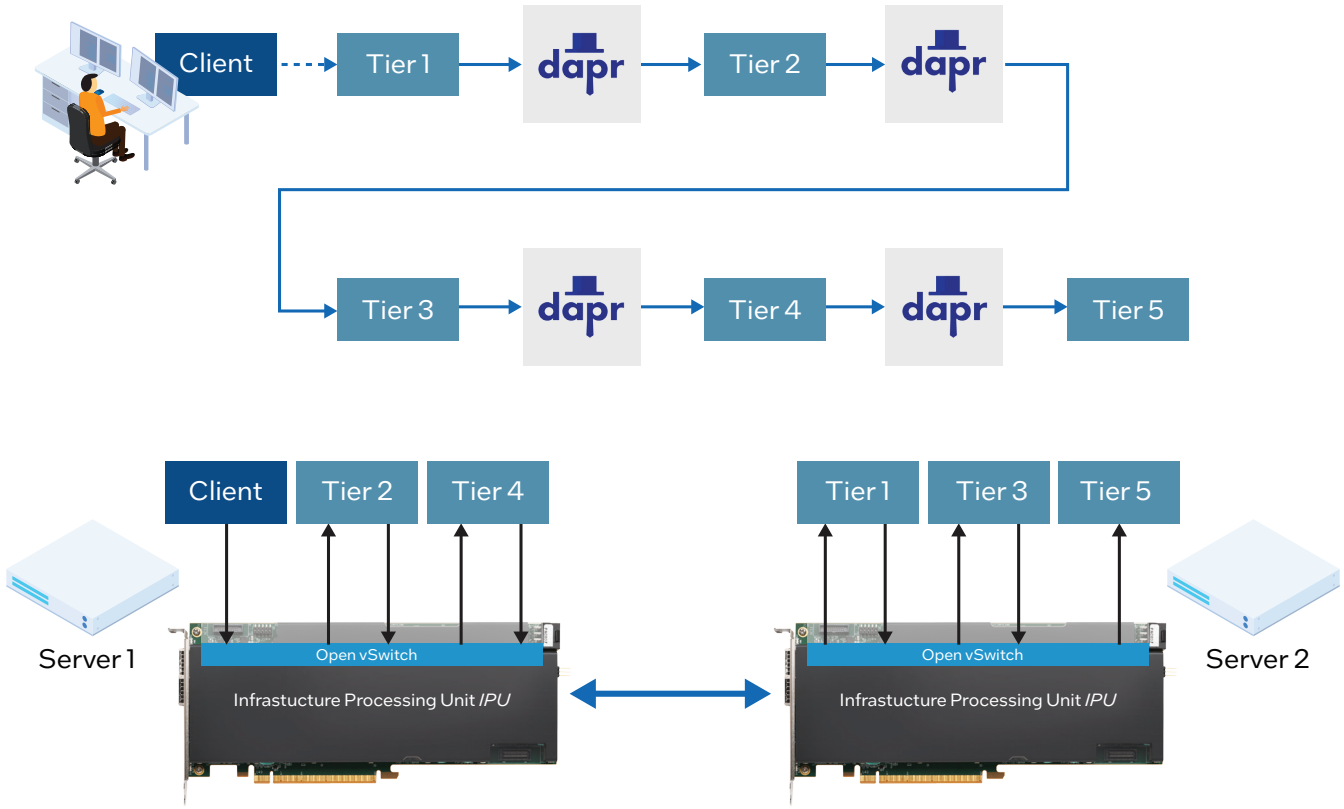


Figure 3. Traffic flow in the IPU-offloaded 5-tier pub-sub application

MIT analyzed the performance of the pub-sub system both with and without the IPU-based offload, measuring the messaging latency across varying loads which are expressed as thousands of queries per second (kQPS).

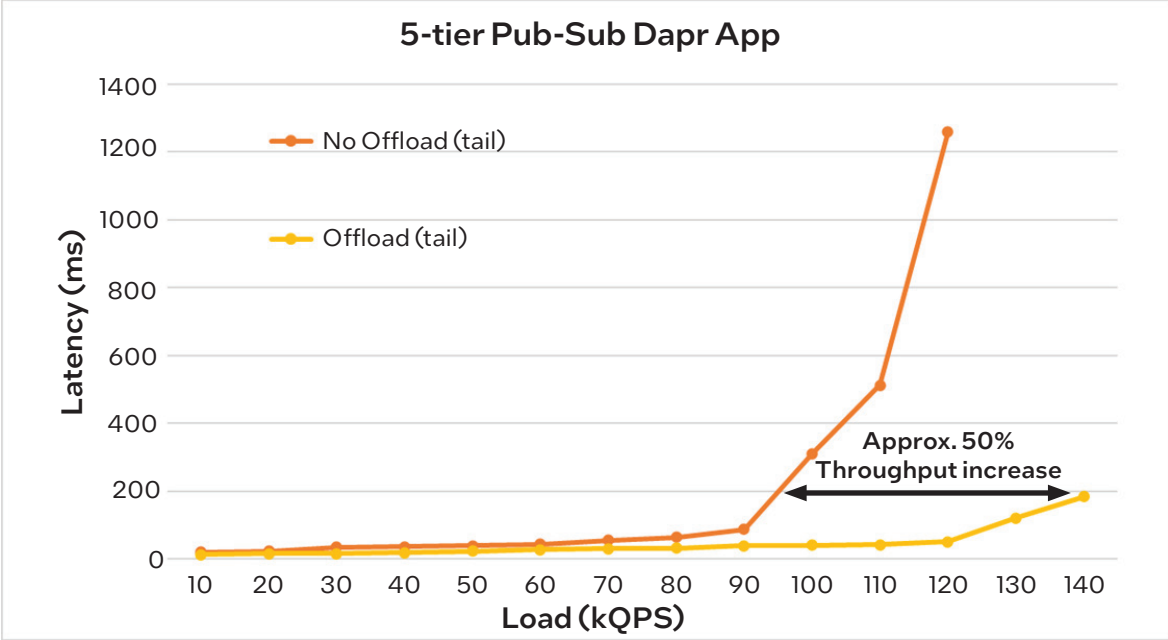


Figure 4. Latency and throughput improvements with IPU offload for 5-tier Pub-sub application

When offload is disabled and considering tail (i.e. worst-case) latency, the application starts to saturate at 90kQPS, as indicated by the inflection point in the graph. Beyond that load level, the system can no longer efficiently keep up with requests, most likely due to packet drops that result in TCP retransmissions. When offload is enabled, however, the system is still keeping up with requests at a load of 140kQPS, the maximum rate used in this test, indicating that **the IPU enables a 50% increase in throughput while maintaining acceptable tail latency**.

This represents a significant improvement in system capacity, resulting in savings of 30-40% in both total server cost and energy consumption.

Three-tier TCP Application Performance

A three-tier TCP (Transmission Control Protocol) application refers to a software architecture design that divides an application into three logical layers or tiers, each responsible for specific functions. These tiers are typically referred to as the presentation tier, application tier, and data tier. The TCP protocol is used for communication between these tiers.

- **Presentation Tier:** Also known as the user interface (UI) tier, this layer is responsible for presenting the application's information to users and receiving their inputs. It deals with graphical user interface (GUI) components, such as web pages, forms, or desktop interfaces. The presentation tier communicates with the application tier to retrieve or update data as necessary.
- **Application Tier:** The application tier contains the business logic and processing logic of the application. It handles the core functionality and performs tasks such as data validation, business rules enforcement, and application-specific operations. This tier processes the requests from the presentation tier and communicates with the data tier to retrieve or store data.
- **Data Tier:** The data tier, also known as the data access layer or database tier, is responsible for managing the storage and retrieval of data. It handles interactions with the database systems, such as querying and updating data. The data tier receives requests from the application tier and returns the requested data or performs the necessary data modifications.

In a three-tier TCP application, the communication between these tiers is facilitated using the TCP protocol. TCP ensures reliable and ordered delivery of data between the tiers, providing a connection-oriented and stream-based communication mechanism. By separating the application into these three tiers, the three-tier TCP architecture allows for modularity, scalability, and easier maintenance of the application. Each tier can be developed and scaled independently, facilitating flexibility and reusability of components.

For this analysis, MIT evaluated a three-tier application with NGINX as the front-end web server, Memcached as the in-memory caching tier, and MongoDB as the back-end database with persistent storage. Clients interact with NGINX, which checks if a key-value pair is cached in Memcached and, if so, returns the value to the client. If not, NGINX interfaces with MongoDB to fetch the output and additionally cache it in memcached.

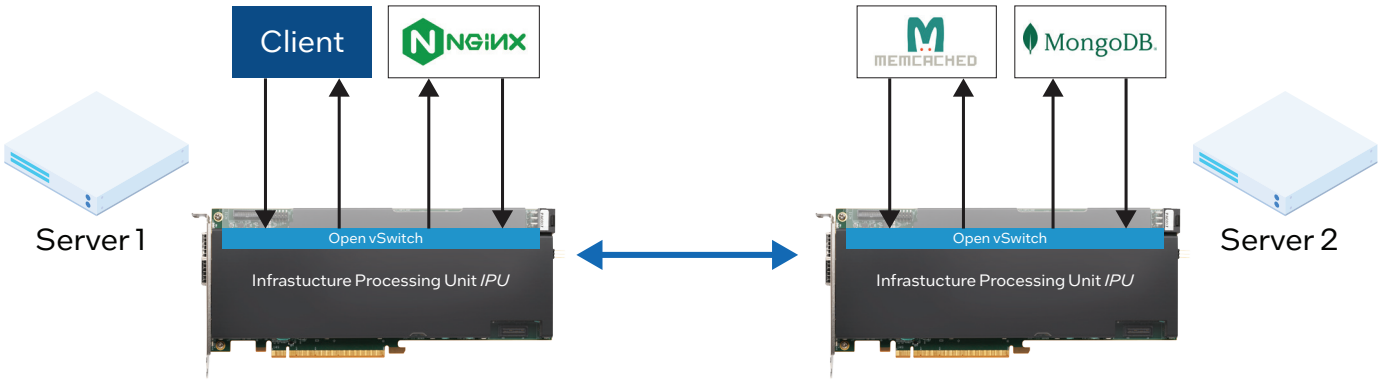


Figure 5. Traffic flow in the IPU-offloaded 3-tier TCP application

MIT analyzed the performance of the three-tier TCP application both with and without the IPU-based offload, measuring the messaging latency across varying loads which, as in the previous example, are expressed as thousands of queries per second (kQPS).

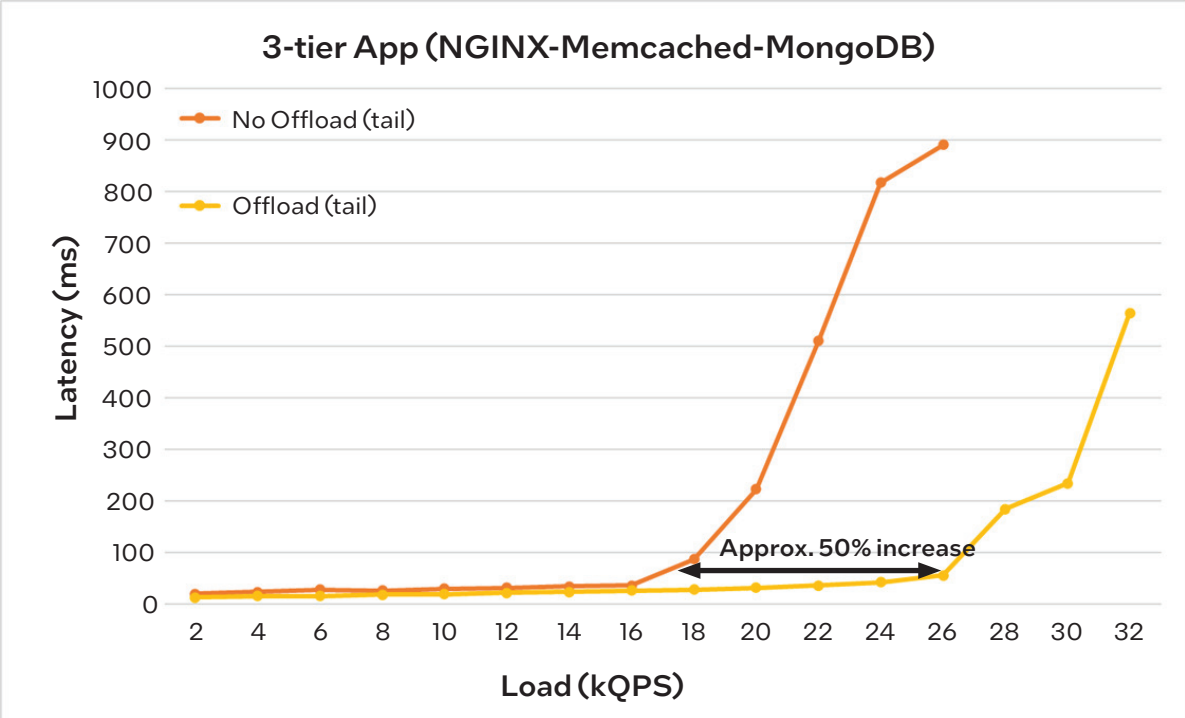


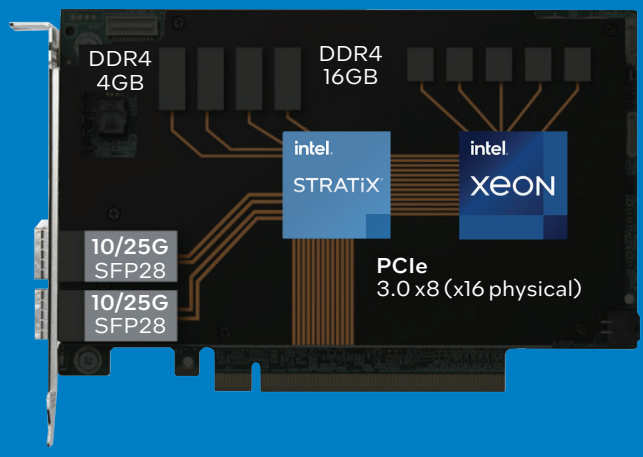
Figure 6. Latency and throughput improvements with IPU offload for 3-tier TCP application

When offload is disabled and considering tail (i.e. worst-case) latency, the application starts to saturate at approximately 17 kQPS, as indicated by the inflection point in the graph. Beyond that load level, the system can no longer efficiently keep up with requests, most likely due to packet drops that result in TCP retransmissions. When offload is enabled, however, saturation does not start until a load of 26kQPS, indicating that **the IPU enables a 53% increase in throughput while maintaining acceptable tail latency**.

Like the previous example, this represents a significant improvement in system capacity, resulting in savings of 30-40% in both total server cost and energy consumption.

The system configuration used by MIT for microservices benchmarking was as follows:

- Two **Inspur dual-socket servers**, each featuring an Intel® Xeon® Gold 6338 Processor with 48MB cache, running at 2.0 GHz with 3.2 GHz turbo speed. Each server was configured with 512GB memory, a 480GB boot drive, dual 1.6TB P6410 NVMe storage modules and one 10G Intel® Ethernet Controller XL710 NIC.
- In addition to the standard NIC, each server was configured with one Intel IPU adapter C5000X with dual 10/25G SFP28 ports and a PCIe 3.0 host interface, based on an Intel® Stratix® FPGA and Intel® Xeon® D System-on-Chip (SoC).
- Each IPU was running the **Link-Virtualization 4.3.3 software** from Napatech, providing an offloaded and accelerated virtualized data plane including functions such as Open vSwitch (OVS), VirtIO support, live migration, VM-to-VM mirroring, VLAN/VxLAN encapsulation/decapsulation, Q-in-Q, RSS load balancing, link aggregation and Quality of Service (QoS).



Summary

Microservices-based software architectures deployed by cloud service providers deliver important benefits such as accelerated deployment, simplified debugging, and improved scalability. However, they introduce significant networking overheads, so that parameters such as network latency have a major influence on overall application performance and data center cost.

By leveraging Intel FPGA IPUs running virtualized data plane software from Napatech, service providers can maximize the performance of their networking infrastructure, enabling a level of performance otherwise unachievable while minimizing their overall data center CAPEX and OPEX.

An analysis performed by MIT demonstrated that in two typical microservices-based use cases, **this IPU solution enables a 50% increase in system throughput** compared to a system configuration based on a standard Network Interface Card (NIC). This enables service providers to decrease the number of servers required to support their total workload for these use cases by approximately one-third, driving a significant reduction in server CAPEX, OPEX, and energy efficiency.

For more information

Detailed information on the Intel IPU Platfrom C5000X is available here and details of Napatech's Link-Virtualization software for this platform are here. The MIT paper with a full performance analysis of the two microservices use cases is here.