



<https://hps.vi4io.org>

Zoya Masih

Performance Estimation

Base on slides from Julian Kunkel

Table of Contents

1 Introduction to Performance

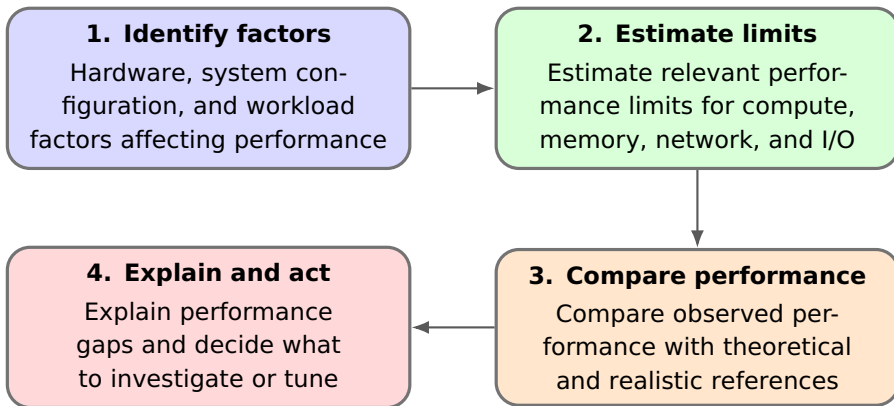
2 Performance Factors

3 Assessing Performance

4 Example: Parallel FS

5 Summary

The Big Picture



Outline

- 1 Introduction to Performance
- 2 Performance Factors
- 3 Assessing Performance
- 4 Example: Parallel FS
- 5 Summary

Introduction

Performance estimation helps us judge whether observed performance is reasonable for our system

Introduction

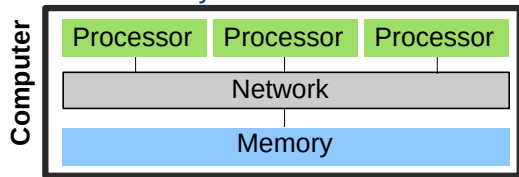
Performance estimation helps us judge whether observed performance is reasonable for our system

- Admins must know basic performance aspects to design suitable systems
 - ▶ Capacity planning (how many servers are needed)
 - ▶ Optimizing systems (higher efficiency)
- System perspective goal:
 - ▶ Efficiency: Better utilization of hardware can reduce the resources required
 - ▶ Simple deployment and easy management
 - ▶ Balance perf with operational requirements i.e security, privacy, compliance
- User perspective goal:
 - ▶ Minimal time to solution, easy to use

Parallel & Distributed Architectures

Different architectures expose different performance challenges

Shared memory

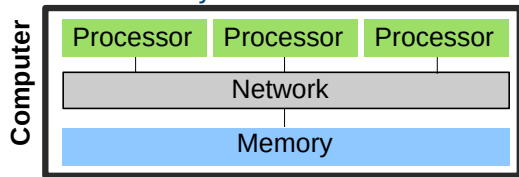


- Cores share and compete for mem bw
- Mem BW may limit Performance
- NUMA locality & cache behavior matter
- Adding cores doesn't guarantee proportional speedup

Parallel & Distributed Architectures

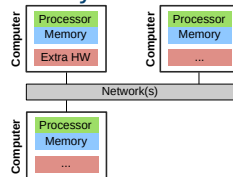
Different architectures expose different performance challenges

Shared memory



- Cores share and compete for mem bw
- Mem BW may limit Performance
- NUMA locality & cache behavior matter
- Adding cores doesn't guarantee proportional speedup

Distributed memory



- Nodes communicate through interconnect
- Latency matters for frequent small message
- BW matters for large data transfers
- Communication & load imbalance limit scaling

Influencing Factors of Performance

1. Hardware

- CPU/GPU compute capabilities
- Memory bandwidth & cache hierarchy
- Interconnect & storage throughput

2. Configuration

- Kernel, drivers, & NUMA policy
- Provisioning choices (*Warewulf*)
- File system & scheduler limits

3. Workload

- Access patterns & compute intensity
- Thread/process placement
- Communication frequency

Admin Key Takeaway

Observed performance results from the interaction of hardware capabilities, system configuration, and workload behavior

Hardware Performance: Relevant Limits

■ Computation / Processing

- ▶ CPU/GPU capability can limit app/sys processing
- ▶ Example: frequency $\times$ cores $\times$ sockets gives a rough aggregate cycle rate

■ Memory

- ▶ Memory bandwidth limits data movement inside clients and servers
- ▶ Example: bandwidth per memory channel $\times$ number of channels

■ Network

- ▶ Bandwidth limits large transfers
- ▶ Latency matters for small/frequent requests
- ▶ Example: 100 Gb/s $\approx$ 12.5 GB/s

■ Storage

- ▶ Device and server throughput limit achievable I/O performance
- ▶ Example: HDDs often provide $\sim$ 100–250 MB/s, NVMe SSDs several GB/s

System Configuration: Administrative Controls

CPU & Memory Controls

- **Power/Frequency Policies:** CPU frequency and idle-state settings affect performance and latency
- **NUMA Placement:** CPU & memory placement affect data locality & remote-memory access
- **Memory Configuration:** Page size and memory policies can affect memory-access overhead

I/O & Network Controls

- **Storage Configuration:** I/O scheduling, file-system settings, and striping affect I/O behavior
- **Network Configuration:** MTU, buffers, interfaces, and bonding affect communication
- **Resource Controls:** Slurm, cgroups, and system limits determine available resources

Memory Access Pattern Matters

Row-wise (Cache-Friendly)

```
1 for (int i = 0; i < N; i++)  
2   for (int j = 0; j < M; j++)  
3     sum += A[i][j];
```

- **Sequential:** Consecutive memory accesses
- **Cache Lines:** Good spatial locality
- **Prefetching:** HW prefetching is often effective
- **Result:** Often higher effective memory bandwidth

Column-wise (Strided access)

```
1 for (int j = 0; j < M; j++)  
2   for (int i = 0; i < N; i++)  
3     sum += A[i][j];
```

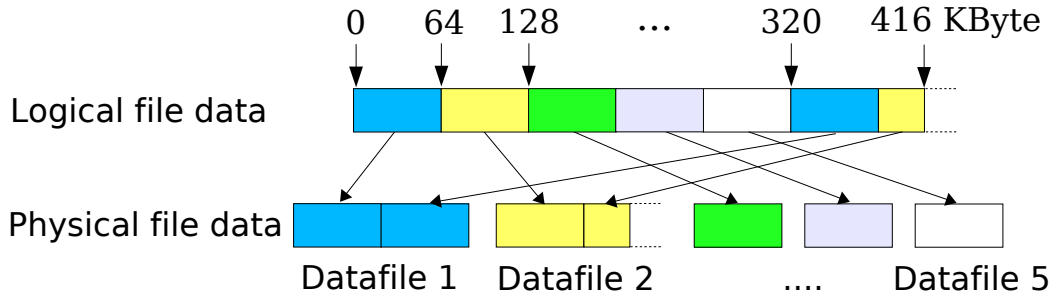
- **Strided:** Jumps between memory locations
- **Cache Misses:** Poorer use of fetched cache lines
- **Prefetching:** May be less effective
- **Result:** Often lower effective memory bandwidth

Sysadmin Takeaway

Peak memory bandwidth is only a ceiling; access pattern strongly affects the bandwidth actually achieved.

Example: Parallel File System

- **Workload:** Multiple clients read/write data through a shared parallel FS
- Files/data are distributed across multiple storage servers
- Aggregate performance depends on clients, network, servers, and storage



Strategy: Is the Observed Performance Reasonable?

Is the observed performance *reasonable for this workload on this system?*

Strategy: Is the Observed Performance Reasonable?

Is the observed performance *reasonable for this workload on this system*?

- Identify the relevant hardware/system limits

Strategy: Is the Observed Performance Reasonable?

Is the observed performance *reasonable for this workload on this system*?

- Identify the relevant hardware/system limits
- Start with a simple estimate of expected performance
 - ▶ Even a rough estimate can reveal a large performance gap
- Refine the estimate only if it cannot explain the observation

Systematic Investigation

1 Observe the symptom

- ▶ Start with system-wide evidence
- ▶ Compare activity across nodes and over time
- ▶ What happened during the workload?
- ▶ CPU/GPU utilization, memory pressure, network traffic, I/O throughput
- ▶ Long runtime, poor scaling, low throughput, high wait time

2 Compare observed behavior with what is expected

- ▶ Is the result close to a theoretical or realistic reference?
- ▶ Can workload or system configuration explain the observation?

Measurements: From System View to Application View

3 Identify the likely resource or system component

- ▶ Compute, memory, network/communication, or I/O

4 Increase measurement detail when needed

- ▶ System and job accounting/metrics: resource use over the job's lifetime
- ▶ Application profiling or tracing: where time is spent inside the workload
- ▶ Targeted benchmarks: isolate CPU, memory, network, or storage perf

Monitoring shows the symptom; deeper measurements help locate its source

Performance Estimation

1. Observe

Quantify the achieved performance

$$P_{\text{obs}} = \frac{\text{Workload}}{\text{Time}}$$

Application / benchmark
Runtime / monitoring data

2. Reference

Theoretical ceiling
Hardware capabilities
Realistic baseline

Comparable benchmarks, prior runs

$$P_{\text{ref}}$$

3. Compare

Compare P_{obs} with P_{ref} to calculate relative performance

$$R = \frac{P_{\text{obs}}}{P_{\text{ref}}}$$

Is the gap significant?
What should be investigated?

Use a theoretical limit for a quick check; use a realistic baseline before judging performance.

Case Study: Object Storage Throughput Analysis

Measured Scenario

- **Data Size:** 100 MiB
- **Measured Time (T):** 0.1 s
- **Observed Throughput (P_{obs}):**

$$P_{obs} = \frac{100 \text{ MiB}}{0.1 \text{ s}} = 1000 \text{ MiB/s}$$

Initial Hardware Ceiling (P_{ref})

- **Network Link:**
100 GbE = 12.5 GB/s
- **Relative to Link Peak:**

$$R = \frac{\approx 1.0 \text{ GB/s}}{12.5 \text{ GB/s}} \approx \mathbf{8\%}$$

Sysadmin Analysis

The workload reaches only about 8% of the network-link peak. This tells us that the 100 GbE link is not saturated, but it does not yet identify the limiting component. Storage, the object-storage service, protocol overhead, request characteristics, or another part of the system path may impose a lower

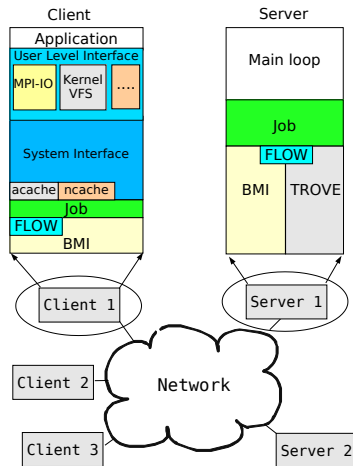
Approximation: Simple Example on Computation

Example:

- **Workload:** summing 10 million integers on the same 3.7 GHz CPU
- Python (for loop): $0.39\text{ s} \rightarrow (10 \cdot 10^6) \cdot 4\text{ bytes} / 0.39\text{ s} \approx 98\text{ MiB/s}$
CPU Cost: $\text{cycles/op} = 3.7 \cdot 10^9 \cdot 0.39\text{ s} / (10 \cdot 10^6) \approx 144\text{ cycles/op}$
- Python (sum()): $0.14\text{ s} \rightarrow \approx 272\text{ MiB/s}, \approx 52\text{ cycles/op}$
- NumPy: $0.0055\text{ s} \rightarrow \approx 7000\text{ MiB/s}, \approx 2\text{ cycles/op}$

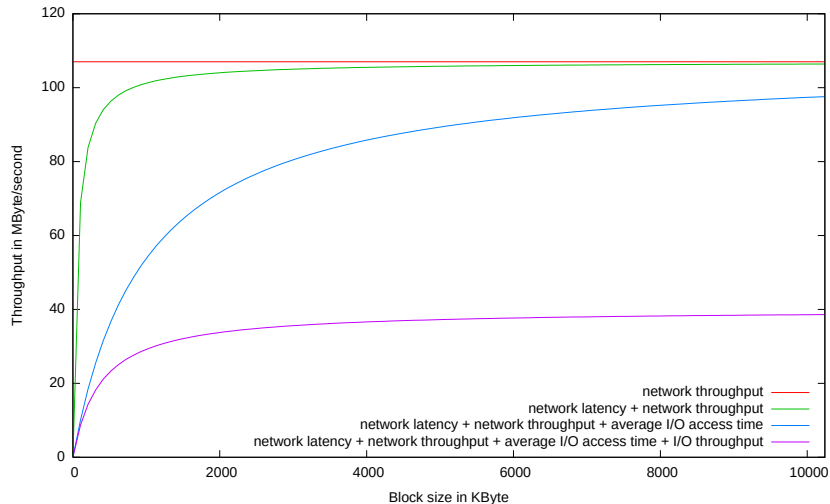
```
1 timeit.timeit(  
2     stmt="np.sum(d)",  
3     setup="import numpy as np; d = np.array(range(1,10*1000*1000))", number=1)  
4 # Compare: sum(range(1,10*1000*1000))
```

Parallel File System Architecture: Here PVFS2

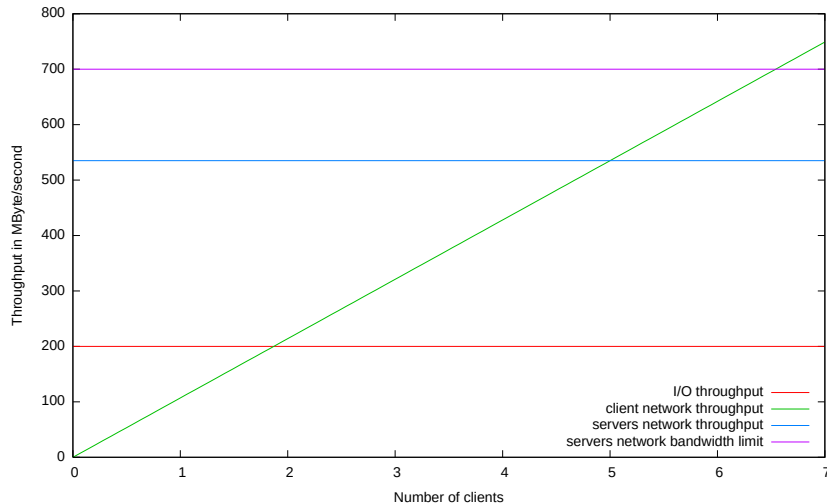


- We can ignore the layers
- C clients connect to S servers
- Clients may access the same file Concurrently - at the same time
- System: GbE Ethernet, HDDs with 40 MiB/s
- Let's build performance models! Start from a simple model and refine

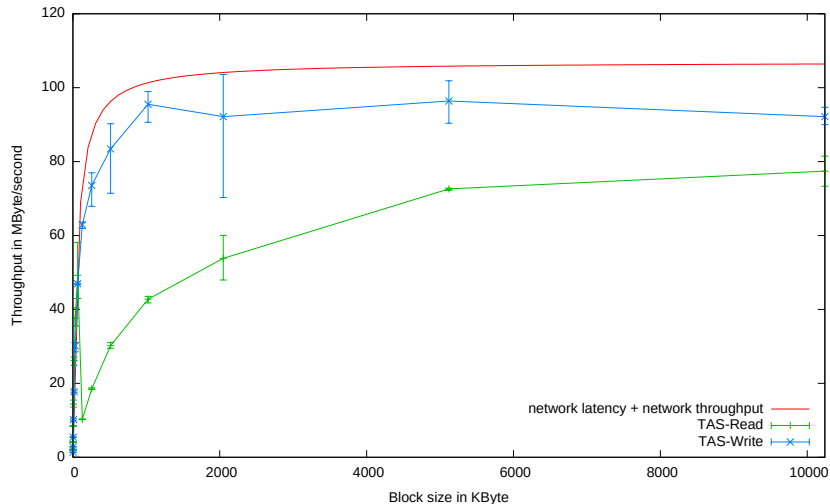
Small I/O Access (Single Client)



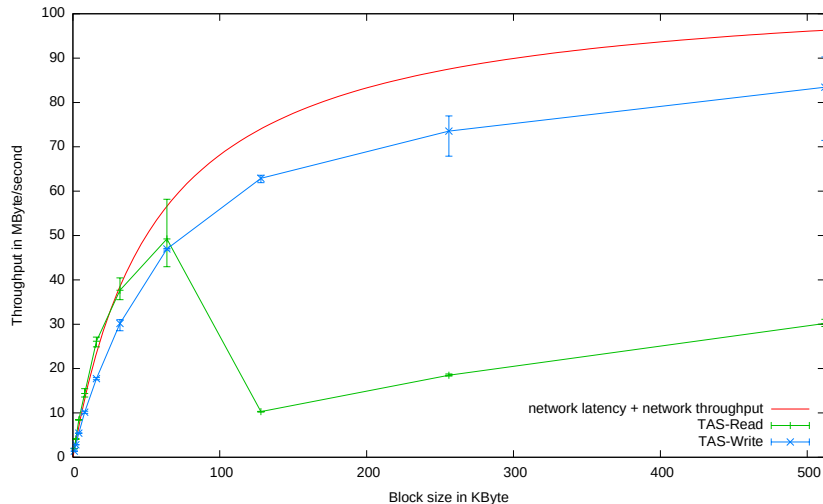
Large Access (Multiple Clients)



Actual Measured Performance (Single Client)



Actual Measured Performance (Single Client) - Small Block Sizes



Summary

- Performance results from the interaction of HW, configuration, and workload
- Start from the question: is the observed performance reasonable?
- Identify the relevant system components and performance limits
- Start with a simple estimate; refine it only when needed
- Compare observed performance with theoretical and realistic references
- A performance gap is a signal to investigate – not proof of a bottleneck
- Use monitoring and targeted measurements to investigate the remaining gap