Cloud Computing

BigTable and provisioning

The notes for the lesson “BigTable and provisioning”: chapters 4.3 to 4.4, with their figures, worked examples, exam traps, quizzes and exam question bank, in one file.

2 chapters | Chapters 4.3 to 4.4

About these notes

These are the notes for the University free n easy video lesson **BigTable and provisioning** on Cloud Computing: chapters 4.3 to 4.4, with every figure, worked example, exam trap and quiz, and the long-answer exam questions on these chapters.

4.3 BigTable and cloud databases

BigTable is a sparse, distributed, sorted map: most cells are empty, rows sit in key order, and data is spread over thousands of machines. A cell is addressed by row key, column key and timestamp. It exists because a web index is too large for one relational database and needs range scans. The NoSQL cloud databases in this unit generalise the trade: they drop joins and fixed schemas to scale out. They run search indexes, click logs, user profiles and time-series data. Some syllabi call this unit 'key-value and semi-structured data storage'.

You need: 4.1 for chunks and metadata, and 4.2 for the replicated file system under BigTable.

Rows, families, qualifiers and versions

BigTable stores an untyped byte string in every cell. A cell is addressed by a row key (a string), a column key, and a timestamp (a 64-bit integer). The column key is written family:qualifier. The row key is the only index. Rows are sorted lexicographically, so keys with the same prefix sit together and a range scan reads them in one pass.

Column families are created before data is written. A family is the unit of storage, compression and access control; its columns are stored together. Inside a family, qualifiers can appear freely, so contents:html and anchor:cnnsi.com need no schema change. A missing cell takes no space; that is what sparse means.

Each cell may hold several versions. Every write gets a timestamp; a read at time T returns the newest version with timestamp at or before T. Old versions are removed by garbage collection rules set per family.

value = map[row][family:qualifier][timestamp]

row is the row key string; family:qualifier is the column key; timestamp is the 64-bit version number; value is an uninterpreted byte string.

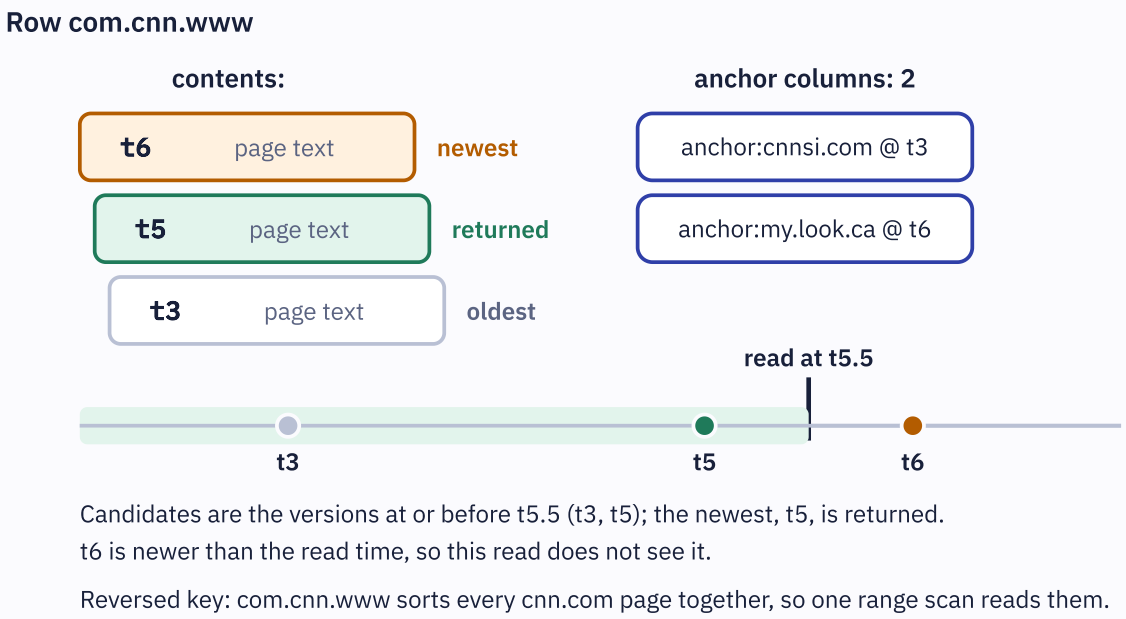


Figure 4.3.1 The row has 3 contents versions and 2 anchor columns. A read at t5.5 looks only at versions at or before t5.5 and returns the newest, t5; the t6 version is not visible to it.

Tablets, METADATA and the write path

BigTable splits a table by row-key range into tablets. A tablet is the unit of distribution. One tablet server owns each tablet; as data grows a tablet splits, and as servers fail tablets move.

A three-level METADATA tree finds tablets. The root tablet never splits. It lists metadata tablets; each metadata tablet lists user tablets and their ranges. A lock service called Chubby holds the root tablet location and detects failure. Clients cache the path, so most reads go straight to the owning server.

A write is durable before it is visible. Steps:

1. Client sends the write to the tablet server for that row range.
2. Server appends the write to a commit log on disk.
3. Server inserts the write into the memtable, a sorted in-memory map.

4. A full memtable is frozen and written to disk as an immutable SSTable.
5. Compaction merges SSTables and removes old versions only when the family's garbage-collection rules allow, for example beyond a version count or an age limit.

An SSTable is an immutable sorted file of key-value pairs. A read merges the memtable and SSTables and returns the newest visible version; bloom filters skip files that cannot hold the key. Replication and recovery targets are in 4.6.

The NoSQL family and consistency choices

BigTable is one point in the NoSQL family. NoSQL means 'not only SQL': keep a simple data model and scale by partitioning and replication instead of joins. Common types:

TYPE	EXAMPLE	DATA MODEL
Key-value	DynamoDB	A key maps to one opaque value; the value may be a document.
Wide-column	Cassandra	Rows with named columns grouped into families; sparse and sorted by row key.
Document	MongoDB	A JSON-like document per key; fields can differ between documents.
Graph	Neo4j	Nodes and edges; traversal follows relationships.

These stores hold semi-structured data: each record carries its own field names, so the schema is flexible. The cost is no planned join across records; the application must model each query into the key.

Consistency is a design choice. Strong consistency returns the latest committed write. Eventual consistency lets replicas converge later, so a read may return an older value. During a network partition a system may choose availability over consistency; that is the CAP trade. A shopping cart may prefer availability; a bank ledger may prefer strong consistency.

Row-key design decides performance

Rows are sorted by key, so the row key is the physical layout. A good key turns the common query into a short range scan; a bad key makes every write hit one tablet.

A crawler stores www.cnn.com as com.cnn.www. If keys kept hostnames unreversed, pages from different cnn.com subdomains such as www, sports and money would sort apart. Reversing the domain components puts all cnn.com hosts under the prefix com.cnn. A range scan then reads those rows contiguously; a large site spans several adjacent tablets.

Monotonic keys create a hotspot. If the key is an always-increasing timestamp, every new write goes to the last tablet while other servers idle. Add a hash prefix or reverse the timestamp to spread writes. Choose the key before the database; no index fixes a bad key.

This is why cloud databases suit large logs, click streams, sensor readings and user profiles. Each query is answered from a key range, not a join. You gain scale and predictable range scans; you give up ad-hoc joins and a rigid schema.

WORKED EXAMPLE

Question. A crawler stores pages under reversed domain names. Row key com.cnn.www has column family contents with three versions at timestamps t3, t5 and t6, and two anchor columns: anchor:cnnsi.com at t3 and anchor:my.look.ca at t6. (a) Which contents value does a read at time t5.5 return? (b) Why reverse the domain? Give the version count and anchor-column count.

1. Contents versions sorted: t3, t5, t6.
2. Candidate versions with timestamp ≤ t5.5: t3 and t5; t6 is excluded because 6 > 5.5.
3. Newest candidate: t5. BigTable returns that contents value.
4. Contents version count = 3; anchor-column count = 2 (anchor:cnnsi.com, anchor:my.look.ca).
5. Reversing www.cnn.com to com.cnn.www makes all cnn.com rows share the prefix com.cnn, so they sit together in sorted order and a range scan reads them contiguously.

Answer. The read returns the contents version stamped t5; the row has 3 contents versions and 2 anchor columns.

EXAM TRAPS

- Treating BigTable as relational: it has no joins, no foreign keys and no cheap full scan; row-key design is the query plan.
- Using increasing timestamps as row keys; every write lands on the last tablet and creates a hotspot.
- Confusing a column family with a qualifier; families are schema and storage units, qualifiers are free names inside a family.
- Reading a timestamp as a clock display; it is a version number, and a read returns the newest version at or before its time.
- Mixing strong and eventual consistency; eventual reads may return an older value after a write.

CHECK YOURSELF

– REMEMBER **Name the three parts of a BigTable cell key.** 2 marks

Row key, column key in the form family:qualifier, and timestamp.

– UNDERSTAND **Why are column families created before data, while qualifiers are not?** 3 marks

A family defines the storage, compression and access-control unit, so it must exist in the schema. Qualifiers are names inside a family and can appear on first write.

– APPLY **Row keys are increasing timestamps. Which tablet gets new writes, and what is one fix?** 3 marks

Writes go to the tablet holding the largest row key, the last tablet. Add a hash prefix or reverse the timestamp to spread writes.

– ANALYZE **Contents versions are t3, t5 and t6. A read arrives at t5.5. Which version returns, and why not t6?** 4 marks

Return t5. BigTable returns the newest version with timestamp ≤ the read time; t6 is newer than t5.5, so it is not visible.

– APPLY **A scan reads keys com.cnn.www, com.cnn.sport and com.cnn.news. How many row-key ranges does it touch if tablets split at com.cnn?** 3 marks

One range: all keys start with com.cnn and sit together after sorting, so one tablet range can serve the scan.

– UNDERSTAND **Give one difference between strong and eventual consistency.** 3 marks

Strong consistency returns the latest committed write on every read; eventual consistency may return an older value until replicas converge.

– ANALYZE **Why does a document store fit semi-structured data better than a fixed relational table?** 4 marks

Each document carries its own field names, so records with different fields coexist without null columns or schema migration. The trade is that joins across documents are not planned by the database.

– REMEMBER **Name the four NoSQL types in this chapter.** 2 marks

Key-value, wide-column, document, graph.

BEYOND THE SYLLABUS

Not for the exam. BigTable's design lives on in managed wide-column services and in systems that combine strong consistency with global scale. The storage engine idea to search next is the LSM-tree: a memtable plus sorted immutable files plus compaction. The consistency debate to search next is CAP and its practical cousin PACELC. Modern cloud databases add serverless scaling and per-request pricing, which change how you choose a row key because cost follows the request pattern. Search next for LSM-tree, CAP theorem, DynamoDB partition keys, and Cassandra compaction.

4.4 Resource provisioning and allocation

Resource provisioning decides how much capacity a provider makes available; resource allocation decides which host and which slot actually serves each request. Both are judged by two numbers: utilisation, the share of installed capacity doing useful work, and the SLA, the promise made to the tenant. Give too much capacity and money burns; give too little and the promise breaks. Every provider runs this loop, from a UPI app on a Friday evening to a Chennai startup's batch job at 2 a.m., and simulators use hosts, VMs and MIPS ratings to study it.

You need: virtual machines and live migration from an earlier lesson, hypervisors; SLAs and automated scaling from an earlier lesson, elastic and on-demand computing.

The two decisions, kept apart

Provisioning answers *how much*. Allocation answers *where*. Provisioning runs on a slow clock: a monthly capacity plan, an autoscaling rule firing in minutes. Allocation runs fast: a scheduler places a VM or container in seconds, and live migration may move it later.

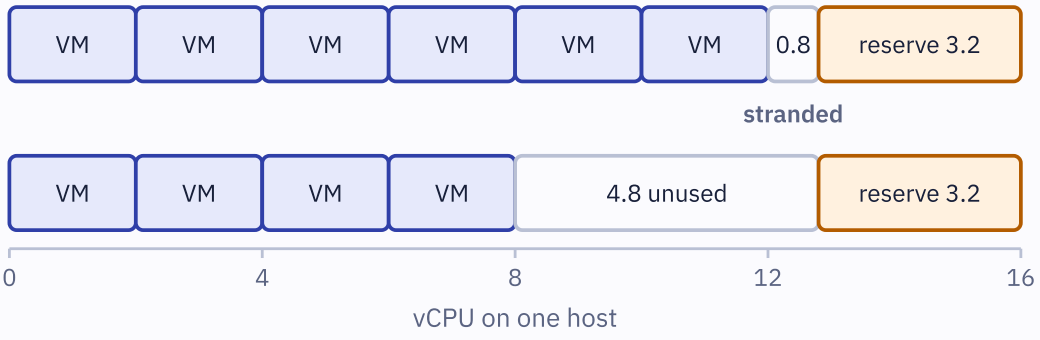
Admission control sits before both. It checks the request against the tenant's quota and the SLA — say 2 vCPU and 8 GB for a new web tier — and accepts, queues or rejects. A provider that accepts work it cannot place has already broken its SLA.

Four resources must fit at once: vCPU (a virtual processor core), RAM in GB, disk IOPS (input/output operations per second) and network bandwidth in Mbit/s. A host with free CPU but a saturated disk fits nothing more. Allocation is therefore a vector test, not one number.

```
host accepts request  ⇔  for every resource r:  used_r + need_r ≤ capacity_r
```

used_r is the capacity already promised on that host for resource r; need_r is what the new VM asks for; capacity_r is the host's total for that resource. If any r fails, the host is rejected.

10 VMs of 2 vCPU on hosts of 16 vCPU (20% kept for the hypervisor)



CPU limit ⌊12.8 ÷ 2⌋ = 6 VMs, RAM limit ⌊64 ÷ 8⌋ = 8 VMs: the smaller, 6, wins. 10 VMs need 2 hosts.

Allocated 20 of 32 vCPU = 62.5%. Idle 12 = 6.4 reserve + 0.8 stranded + 4.8 unused.

Figure 4.4.1 CPU allows 6 VMs per host and RAM 8, so 6 wins and 10 VMs take 2 hosts. Of the 32 vCPU, 20 are allocated (62.5%); the 12 idle are 6.4 reserve, 0.8 stranded on host 1 and 4.8 unused on host 2.

How much: static, dynamic and headroom

Static provisioning reserves a fixed slice — a rack, a committed 100 Mbit/s line. Easy to bill, wasteful when idle. Dynamic (elastic) provisioning adds and removes capacity with demand: a ticket site before a final, a retail app on a sale day. It needs a monitor, a scaling rule (add a VM when mean CPU over 5 minutes passes 70 percent) and a cooldown so it does not oscillate.

Over-provisioning buys safety and wastes money. Under-provisioning saves money and breaks the SLA through throttling, long queues or out-of-memory kills. Both are read off utilisation:

```
U = (busy capacity ÷ installed capacity) × 100%   and   A = (promised capacity ÷ ins
```

U is utilisation in percent; busy capacity is what is actually doing work. A is the allocation ratio in percent; promised capacity is what tenants were given. installed capacity is what the provider bought and powered. The worked example and quizzes below report A, the allocation ratio, because busy load is not given.

Keep headroom. The hypervisor needs CPU and RAM of its own, so planning often holds back 10 to 20 percent of vCPU. A second reserve stops a host thrashing: if RAM is sold to 100 percent, the hypervisor swaps and every VM on that host slows down. Never plan to 100 percent.

In simulators a host is rated in **MIPS**, millions of instructions per second. That is an instruction rate, not a clock speed in MHz. Each VM or cloudlet asks for a share of the host's MIPS, and the simulator checks that the shares sum to no more than the rating.

Which host: allocation as bin packing

Placement is bin packing. Hosts are bins, VMs are items, and item sizes have several dimensions. Bin packing is NP-hard, so real schedulers use fast heuristics.

1. Sort pending VMs largest first (first-fit decreasing).
2. Walk the host list in order.
3. Place the VM on the first host where every resource fits.
4. If no host fits, open a new host or queue the request.

Best-fit picks the host with the least free space left after placing, which packs tightly and reduces the number of hosts used, but it leaves small leftover gaps that a later large VM cannot use. First-fit is faster and leaves larger free regions on the later hosts. Worst-fit spreads load, useful when VMs are bursty and share a network link.

Rules override the packing. Affinity keeps two VMs together, such as a web tier and its cache, to cut latency. Anti-affinity keeps them apart, such as two replicas of a database, so one host

failure cannot take both.

Live migration rebalances without downtime: copy a running VM's memory pages to another host, then switch over. Use it on hot hosts, to drain a host before maintenance, and after a scale-out. It spends network bandwidth, so it is not free.

WORKED EXAMPLE

Question. A host has 16 vCPU and 64 GB RAM. The provider keeps 20 percent of the vCPU for the hypervisor and does not oversubscribe RAM. Each VM requests 2 vCPU and 8 GB RAM. (a) How many VMs fit on one host? (b) How many hosts are needed for 10 VMs? (c) What is the CPU utilisation (allocation ratio) of that cluster? (d) How many vCPU sit idle, and why?

1. Usable vCPU = 16 × (1 − 0.20) = 12.8 vCPU.
2. CPU limit = floor(12.8 ÷ 2) = 6 VMs.
3. RAM limit = floor(64 ÷ 8) = 8 VMs.
4. The smaller limit wins: min(6, 8) = 6 VMs per host.
5. Hosts for 10 VMs = ceil(10 ÷ 6) = 2 hosts.
6. Allocated vCPU = 10 × 2 = 20 vCPU.
7. Installed vCPU = 2 × 16 = 32 vCPU.
8. Utilisation = (20 ÷ 32) × 100 = 62.5 percent.
9. Idle vCPU = 32 − 20 = 12 vCPU: 6.4 vCPU held as hypervisor reserve (2 × 3.2), 0.8 vCPU stranded on the first host because 12.8 − 12 = 0.8 is too small for one more 2-vCPU VM, and 4.8 vCPU simply unused on the second host because only 4 of its 6 slots are filled.

Dry run

HOST	VCPU USED	RAM USED
H1	12	48 GB
H2	8	32 GB

Answer. 6 VMs per host, 2 hosts for 10 VMs, 62.5 percent CPU utilisation (allocation ratio), with 12 vCPU idle (6.4 reserved, 0.8 stranded, 4.8 unused).

EXAM TRAPS

- Dividing host capacity by VM size and stopping there. Every resource must fit, so the minimum over all resources decides the count.
- Swapping the two words: provisioning is how much capacity, allocation is which host. An answer that places VMs when asked about provisioning loses marks.
- Treating MIPS as a clock speed. MIPS is millions of instructions per second, an instruction rate used by simulators, not MHz.
- Planning RAM to 100 percent. Without a reserve for the hypervisor and for burst, the host swaps and every VM on it slows down.
- Forgetting rules: anti-affinity can cut the number of VMs that fit even when free vCPU and free RAM both exist.

CHECK YOURSELF

– REMEMBER **Define resource provisioning and resource allocation in one line each, and say which runs on the slower clock.** 2 marks

Provisioning decides how much capacity to make available; it runs on the slower clock (capacity plans, autoscaling rules). Allocation decides which host serves each request; it runs in seconds.

– UNDERSTAND **A host shows 40 percent free CPU but its disk is at 100 percent IOPS. Explain why a new VM that writes heavily is still rejected.** 3 marks

Allocation is a vector test: used_r + need_r ≤ capacity_r for every resource. Disk IOPS fails, so the host is rejected even though CPU is free. The saturated disk is the bottleneck.

– APPLY **A host has 32 vCPU and 128 GB RAM, with 25 percent of vCPU reserved for the hypervisor. Each VM needs 4 vCPU and 16 GB. How many VMs fit?** 3 marks

Usable vCPU = 32 × 0.75 = 24; CPU limit = 24 ÷ 4 = 6 VMs. RAM limit = 128 ÷ 16 = 8 VMs. Minimum = 6 VMs per host.

– APPLY **Using the host above, how many hosts are needed for 20 VMs, and what is the CPU allocation ratio?** 4 marks

Hosts = ceil(20 ÷ 6) = 4. Allocated vCPU = 20 × 4 = 80; installed = 4 × 32 = 128; allocation ratio = 80 ÷ 128 × 100 = 62.5 percent.

– ANALYZE **Compare first-fit and best-fit placement for a stream of mixed-size VMs arriving over a day.** 4 marks

First-fit scans hosts in order and takes the first that fits: fast, but it fills early hosts and leaves many half-empty ones. Best-fit takes the host with the least space left after placing: tighter packing and fewer hosts, but it scatters small gaps that a later large VM cannot use, and it costs a full scan of the host list.

– ANALYZE **Two database replicas are placed with an anti-affinity rule. Explain the effect on usable capacity and on fault tolerance.** 4 marks

Anti-affinity forces the replicas onto different hosts, so a host that has room for both can take only one. Usable capacity per host falls and more hosts are needed. Fault tolerance rises: one host failure takes at most one replica, so the database stays available.

– UNDERSTAND **Why is a host not filled to 100 percent of its RAM, even when tenants are willing to pay?** 3 marks

The hypervisor needs memory of its own, and a reserve absorbs bursts. At 100 percent the host swaps and every VM on it slows, so throughput and SLA compliance fall below what the extra sale is worth.

– REMEMBER **What does MIPS stand for in a cloud simulator, and what does a host's MIPS rating mean?** 2 marks

MIPS is millions of instructions per second. A host's rating is the total instruction rate it can supply; each VM or cloudlet requests a share of that rating, and the shares must fit within it.

BEYOND THE SYLLABUS

Not for the exam. Kubernetes' scheduler is a live version of this loop: filter hosts that fit, score the survivors by weighted rules, then bind. Research moves toward learned placement, where a model predicts demand from past traces instead of a fixed CPU threshold. CloudSim lets you test a policy against thousands of VMs before renting one. Search for vector bin packing for VM placement, autoscaling cooldown, and host MIPS in CloudSim.

Key terms of this lesson

TERM	MEANING OR FORMULA	CHAPTER
BigTable	A sparse, distributed, sorted map from (row key, column, timestamp) to a byte value, split into tablets and stored on many machines.	4.3
Column family	A named group of columns in BigTable that is stored together and compressed together; you design the table around families, not around individual columns.	4.3
NoSQL	A family of stores (key-value, column, document, graph) that drop joins and fixed schemas to spread data over thousands of machines.	4.3
Provisioning	Deciding how much capacity — how many hosts, VMs, vCPUs, GB of RAM and IOPS — to make available before any request arrives.	4.4
MIPS	Million instructions per second, the unit CloudSim uses to state how much computing power a host gives a VM; a host of 1000 MIPS split 500/500 gives each of two VMs 500 MIPS.	4.4
SLA	Service level agreement: the promised response time, uptime and throughput, with penalties if the promise is broken; it is the limit inside which utilisation may rise.	4.4

Exam question bank

Long-answer exam questions on these chapters, each with an outline of the points an examiner looks for. Answer one in full on paper before you read its outline.

Remember and understand

– UNDERSTAND **Describe BigTable as a data model. Explain row key, column family, timestamp and sparsity, and why rows are kept in sorted order.** 13 marks

Model is a map: (row key, column, timestamp) → byte string; nothing else is assumed about the value; Rows are sorted lexicographically by row key and split into tablets that migrate between servers; Column families are created up front and stored and compressed together; new columns inside a family need no schema change; Timestamps keep several versions of a cell; garbage collection keeps the last n or the last t seconds; Sparsity: absent cells take no space, so a very wide table is normal; Sorted order makes range scans by key prefix cheap — the design reason for reversing the domain in row keys so that related items sit together.

– UNDERSTAND **Distinguish resource provisioning from resource allocation. Define MIPS and show how MIPS is used to decide whether a host can hold one more VM.** 8 marks

Provisioning: deciding total capacity and making it available ahead of time (hosts, VMs, storage, network); Allocation: at request time, choosing which host or VM serves this request; MIPS = million instructions per second; a host has total MIPS and total RAM, and each VM asks for a share; Feasibility test: sum of allocated MIPS ≤ host MIPS and sum of allocated RAM ≤ host RAM, plus bandwidth and storage checks; Oversized provisioning wastes money; undersized provisioning overruns the SLA; the aim is minimum slack consistent with the SLA.

Apply

– APPLY **A service has 10 identical hosts, each carrying 8 VMs of 1000 MIPS. Demand over 24 hours is given in a table. Show how consolidation with live migration reduces the number of powered hosts, and check the SLA after consolidation.** 13 marks

Compute total MIPS demanded in each interval and divide by host capacity to get the minimum host count; Compare with the always-on 10-host baseline: the difference is the idle capacity that consolidation removes; Live migration moves a VM with a short pause; it costs network bandwidth and a little CPU on both ends, so migrate during a trough; Check SLA: after consolidation no host should exceed a safe utilisation ceiling, because queueing delay grows sharply near 100 percent; Compute power saved as (hosts removed) × idle power, then add the saving from DVFS on the surviving hosts; State the risk: too aggressive consolidation leaves no headroom for a sudden spike; keep a reserve.

Glossary (terms used in this lesson)

Allocation	Choosing the host or VM that will actually serve a given request or task.
Chunk	A fixed-size slice of a file that a distributed file system stores, replicates and checksums as one unit.
Cloudlet	In CloudSim, one task with a length in million instructions and a file size, placed in a VM.
Column family	A named set of BigTable columns stored and compressed together, created when the table is created.
Fault tolerance	The property that the service keeps meeting its promise when a component fails, using detection, redundancy and recovery.
Hypervisor	The layer that runs virtual machines and divides physical CPU, memory and I/O among them.
Live migration	Moving a running VM to another host with a pause short enough that users do not notice.
MIPS	Million instructions per second, the capacity unit CloudSim uses for hosts, VMs and tasks.
Provisioning	Making capacity available ahead of demand, in hosts, VMs, storage and network.
Tablet	A contiguous sorted range of BigTable rows that is the unit of distribution and migration.
Virtual machine	A software-defined computer with its own vCPUs, RAM and disk, running on a shared physical host.

Sources and further reading

Textbooks

- Andrew S. Tanenbaum and Maarten van Steen, *Distributed Systems: Principles and Paradigms*: Chapters on distributed file systems, replication and fault tolerance cover most of this unit's architecture material.
- Thomas Erl, Ricardo Puttini and Zaigham Mahmood, *Cloud Computing: Concepts, Technology and Architecture*: Cloud storage, IaaS provisioning, SLAs and cloud delivery models.
- Kai Hwang, Jack Dongarra and Geoffrey Fox, *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*: Resource management, scheduling, energy-aware computing and federation.

- Tom White, *Hadoop: The Definitive Guide*: HDFS internals, HBase (BigTable-style storage) and MapReduce on the cluster.

Open web resources

Each link below was opened by the book script on 11 Nov 2026 and returned a page; the pages were not read line by line.

- Apache Hadoop project and HDFS documentation: hadoop.apache.org
- Google Cloud Bigtable documentation overview: cloud.google.com/bigtable/docs/overview
- Amazon S3 product and documentation page: aws.amazon.com/s3
- Amazon EC2 product and documentation page: aws.amazon.com/ec2
- Kubernetes documentation, Concepts section: kubernetes.io/docs/concepts
- OpenStack project site: www.openstack.org

Syllabus: the University free n easy universal syllabus, built from top universities' syllabi.

Built for the University free n easy YouTube channel from the universal syllabus of Cloud Computing (Oct 2026). Written with top universities' syllabi and standard textbooks; check a formula against your prescribed book before an exam, and report mistakes on the channel.