Computing Infrastructure

Elia Ravella

March 4, 2021

Contents

Part I
Introduction

1 Computing Infrastructure

What is a CI? A CI is a *technological structure* that provides hardware and software for computation to other systems. This means a lot: is a joint (and heterogeneous) system that comprises both HW and SW to deliver a service. With CI is not intended the *application delivered* but the combination of elements that makes the application available and running.
So everything in between a RaspPI that does torrent seeding from a local network to the whole datacenter that runs an AWS service can be considered a computing infrastructure.

Part II
Hardware Infrastructures

2 Data Centers

2.1 Pros and cons

Data centers are big centralized CIs that comprise a lot of servers (to provide computations) a communication infrastructure, and a storage service.
PROS of a DC:

- Lower IT cost: renting a virtual machine is cheaper than buying / building and maintaining a whole system (for some time horizons, of course);
- High performance: virtualized resources make scaling easier, and provides optimized and finely tuned services that an in-house solution cannot provide *so* easily and fast;
- No effort software updates;
- Unlimited (!) storage capacity;
- Increased data reliability;
- Universality of accesses;
- Device indipendence.

The CONS of a DC can be found simply inverting the POV for the PROS aspects: outsourcing resources gives "someone else" the control over some crucial aspects of a CI. This is a good thing when costs (also time costs) must be reduced, but can be a bad thing when a fine grained control over a full system is needed. Also, *latency* pops in, due to the needed connection to the DC.

2.2 DCs ans WSCs

The data center approach has been emerging in the last years. The idea behind it is that we should not "overpack" nodes of a network with all the computing capabilities required, but instead giving them a connection to such CIs. Data centers are the perfect example for this paradigm: the user interact with a client (that can be *any kind of device*) that also interact with a remote structure to provide computations. To a careful watch, SaaS and their spread are a direct conseguence of this paradigm shift. Another use of DCs is using their computing capabilities not in a fragmented way to provide miriads of services, but to perform an *extremely costly computation*, like a training of a neural network.
From a DC approach we are moving to a Warehouse Scale Computers nowadays. This latter approach consists of NOT "mix up the pot" in a DC (so having a lot of heterogeneous technologies in order to achieve different tasks) but instead to "homogeneify" the cloud structure in order to do better optimization of it. Not only in the HW, but also in the SW. To centralize the DC capabilities under a single organization (as often happens) means that clients no more run their application on someone else's hardware, but instead choose from a set of precooked solutions by such vendor/organization. Is this *bad*? WSCs are just a "SaaS - oriented" Data Center Architectures, so *there's no real transition between DCs and WCSs* the only transition is in the number of *virtualiazed layers the client must go through to access and application.*
We can sum up the difference between DCs and WSCs in this way: where DCs are intended as a powerful collection of different servers, WSCs tries to offer a homogeneous interface that can be used also as a single server to such a collection of hardware. Still, if we consider a lerger definition for DCs, WSCs are a type of DCs.

2.3 Architectural Overview of DCs and WCSs

A standard data center building is organized in separated modules, every one dedicated to a specific task. The four main modules types are

- Servers: computations
- Storage
- Networking: intended as the whole communication infrastructure
- Building-integrated systems:
  - Cooling system: often as powerful as the server themselves, fundamental for getting rid of excess heat
  - Power supply: integrated in the building and provided with systems to avoid power shortage
  - Failure recovery: physically realized as a building module in order to be as most fault - tolerant

Servers are organized in racks, blades or tower, and are the classical computational unit usually found in server farms. They could also not have memory attached.

Storage is the crucial long term memory for a CI. Usually built with flash SSDs and ferromgnetic mechanical disks. The memory units must provide high speed I/O capabilities, and also advanced networking power, also at software level (NAS, SAN, DAS).
The networking infrastructure is the backbone of the communication in (and from/to) a data center. Structured hierarchical approach to networking structures and organization are used to ensure security and performace
The building itself is part of the CI.
Next sections will delve into the details of the single modules.

2.4 Servers

Servers are the computational compartment of a WSC. They're organized in racks (shelves) that host the computational units (pizza box computer). Servers are just computers. That's all. They got a MOBO, local primary memory, a CPU, and I/O capabilities. They can be organized as

- Towers: cheap, cooling is easy, upgrade is easy. They're also big, and they provide low density of computational power.
- Blades:
- Racks: literal racks that host the pizza box shaped computational units, and accomodate all the wiring and additional connection or cooling systems. They can host also heterogeneus components, not only standard computational modules. Rack's dimension and geometry is a standard. Rack organization provides better failure handling and simplified cable management, but they're more power demanding (wrt towers) and mainteinance is a mess (mmultiple devices must be maintained at the same time).

Part III
Software Infrastructure

Part IV
Dependability

3 Intro

Dependability represents the *availability performance* of a system. It encompasses

- Reliability
- Availability
- Maintainability

Dependability is approached with a statistical/probabilistic POV due to the high human component in it.
Two functions describes the system: F(t) and f(t), respectively the cumulative function and the fault probabilistic distribution. The former represents the *unreliability* of the system analyzed. So, we can define $R(t) = 1 - F(t)$ as the reliability function.
Probabilistics recall:

$$\begin{aligned} f(t) &\rightarrow \text{ probabilistic distribution} \\ F(t) = \int_0^t f(t)dt &\rightarrow \text{ cumulative function, unreliability} \\ R(t) = 1 - F(t) &\rightarrow \text{ reliability function} \end{aligned} \tag{1}$$

From this we can define the Mean Time To Failure for a component:

$$MTTF = \int_0^\infty t \cdot f(t)dt \tag{2}$$

Where t represents the failure time. We can also define the failure rate:

$$\lambda(t)dt = F(t < T \leq t + dt \,|\, T > t) \tag{3}$$

So the failure rate represents the probability of failure *assuming the component was working the instant before*. This function can be seen as the number of failures in a given interval.

3.1 Failure Rate

Type of malfuntioning:

- Fault: physical defect or software bugs.
- Error: program incorrectness that can result from a fault.
- Failure: nonperformance of some actions that were expected. They can be result of an error.

Reliability as Weibull Failure rate $\lambda(t)$ function has a peculiar function shape: a *bathub* (or long U) shape. The failure rate is high in the starting period and decreases (infant mortality effect) to the costant level during the useful lifetime (costant probability of failures) and it raises again at the end (wear out period).

In fact, reliability follows the *Weibull distribution* so definied: $y(x) = e^{-(\frac{x}{\alpha})^\beta}$. So (due to the $\lambda(t) = f(t)/R(t)$ relation) we obtain

$$\lambda(t) = \frac{\beta}{\alpha}(\frac{t}{\alpha})^{\beta-1} \tag{4}$$

Reliability as exponential We can also model the reliability as an exponential function (so $R(t) = e^{-\lambda t}$) we than obtain

$$\begin{cases} F(t) = 1 - e^{-\lambda t} \\ f(t) = \lambda e^{-\lambda t} \\ MTTF = \int_0^\infty t \cdot \lambda e^{-\lambda t} dt = \frac{1}{\lambda} \end{cases} \tag{5}$$

So we can express the reliability as function of MTTF: $R(t) = e^{-\frac{t}{MTTF}}$. And we all know that in certain conditions (like $\frac{t}{MTTF} << 1$) an exponential function can be approximated to a linear function.

3.2 System level reliability

We can model the system relibility mainly with RBDs: Reliability Block Diagram. These simply outline the operational dependency between components. The main assumption of the RBD is that the failures are *indipendent*, that implies that are no "cascading failures".
It's easy to calculate the overall reliability:

$$\begin{cases} \text{Serial components: } R_s(t) = \prod_{i=1}^n R_i(t) \\ \text{Parallel components: } R_p(t) = 1 - \prod_{i=1}^n (1 - R_i(t)) \end{cases} \tag{6}$$

The MTTF is also altered from the serialization or parallelization of components. In particular, in a serie with n identical components, $MTTF_{serie} = \frac{MTTF}{n}$. The general formula is the one used for parallel resistors:

$$MTTF_{serie} = \frac{1}{\sum_{i=1}^n (\frac{1}{MTTF_i})} \tag{7}$$

For parallel system, things get complicated. The MTTF is calculated from the unreliability, integrating between 0 and ∞. The resulting formula is

$$MTTF_{parallel} = \sum_{i=1}^n (MTTF_i) - \frac{1}{\sum_{i=1}^n (\frac{1}{MTTF_i})} \tag{8}$$

Notice that the second term of that sum is exactly $MTTF_{serie}$ of that configuration. Again, the formula can be simplified if the components are identical: $MTTF_{parallelidentical} = MTTF_i * (\sum_{i=1}^{n-1} (\frac{1}{n-i}))$

Extension of MTTF calculation to complex systems We can apply the calculation of the *reliability* of a complex systems directly with the formulas, but the same cannot be done with MTTF. This is due to the fact that a complex system has not anymore a simple failure rate distribution, and this messes up the MTTF calculation, that's an integral on that function. So, I must pass through the reliability calculation.

3.3 Availability

Introducing MTTR, Mean Time To Repair. This represents the average time required to replace a failed component and "bring the system back up" after a

failure. It can be the restart time (for software module) or the replace time (for an hardware component).
Another interval to be taken into consideration when talking about availability is the Mean Time Between Failure, that's just the sum of $MTTF + MTTR$. Availability is defined as the probability of a system to be up and running at a given istant. So $Av = \frac{MTTF}{MTBF}$ that corresponds to

$$Availability = \frac{MTTF}{MTTF + MTTR} \tag{9}$$

The slight difference between availability and reliability takes into consideration the repairing of a system in the case it goes down. The availability function is calculated with *the same identical formulas* used to calculate the reliability for parallel and serial components.
Talking about MTTF when taking into consideration that components can be *repaired* must be carefully handled: for example, if a parallel system has *interleaving* failures, it never fails enitrely. A way of calculating MTTF for repairable system consists in inverting formula (9), obtaining

$$MTTF = \frac{Av * MTTR}{1 - Av} \tag{10}$$