Arbitrary Fault-Tolerant and Locality-Aware MapReduce

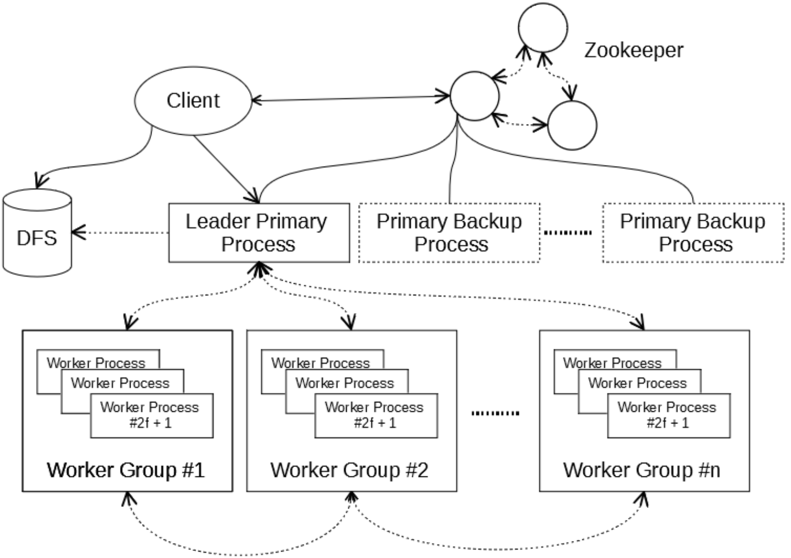
Sistemi Distribuiti e Cloud Computing 2018-2019

Andrea Graziani - 0273395

Università degli Studi di Roma "Tor Vergata"
FACOLTA' DI INGEGNERIA
Corso di Laurea Magistrale in Ingegneria Informatica

2 ottobre 2019

System Architecture



System Architecture

- What assumptions we have made?
- We distinguish three process types:
    - *Client Process*
    - *Primary Process* (PP)
    - *Worker Process* (WP)
- How we made the system fault tolerant? What kind of fault tolerance can be achieved with this design?
- What kind of semantics do we have adopted?
- Is our system scalable?
- Our system can be elastic? **Yes... but**
- What are the advantages/disadvantages of our solution?

System features

Deferred Execution

- By default our system is design to use the *Deferred Execution* approach during task scheduling.
    - What's meant by deferred execution and how it works?
    - What are the advantages of this approach? And the disadvantages?
    - What happens if LPP/WPs crashes?
    - Why *Locality-Aware*?
    - Why *State-Aware*?

System features

Speculative Execution

- In order to improve system's performance we have adopt an approach based on the so called *Speculative Execution*
  - What's meant by speculative execution? How it works?
  - What happens if the input used during Reduce-Phase is detected as incorrect?
  - What happens if LPP/WPs crashes?
- Can we do better? **Yes...but...**

System features

Data-Locality-Aware Reduce Scheduling and Shuffle

- Our system is designed to schedule reduce task considering data locality in order to reduce the overall amount of transferred data between WPs: this approach is called *Data-Locality-Aware Reduce Scheduling*.
    - "*Moving computation towards data is cheaper than moving data towards computation*".
    - How it works?

System features

Other features

- Digest output.
- Leader election and crash failure detection based on Apache Zookeeper.

Grazie per l’attenzione!