



Sistemi distribuiti

Programmazione

I sistemi distribuiti sono architetture informatiche in cui più nodi autonomi (computer, server o servizi) cooperano tra loro tramite una rete per fornire un servizio o risolvere un problema comune, apparendo all'utente come un unico sistema coerente.

A differenza dei sistemi centralizzati, l'elaborazione e la gestione dei dati sono distribuite su più componenti fisicamente separati, che comunicano attraverso protocolli di rete e meccanismi di coordinamento.

Questo approccio consente di migliorare scalabilità, affidabilità e tolleranza ai guasti, ma introduce anche complessità legate alla sincronizzazione, alla latenza di rete e alla gestione della consistenza dei dati.

I sistemi distribuiti sono alla base di molte tecnologie moderne, come il cloud computing, i microservizi e le applicazioni web su larga scala, rappresentando un pilastro fondamentale dell'informatica contemporanea.



Le quattro caratteristiche principali dei sistemi distribuiti possono essere sintetizzate e spiegate come segue:

- **Trasparenza**

Un sistema distribuito deve apparire all'utente come un unico sistema coerente, nascondendo la complessità della distribuzione. La trasparenza riguarda aspetti come l'accesso alle risorse, la loro posizione fisica, la replica dei dati e la gestione dei guasti, permettendo di utilizzare il sistema senza conoscere dove o come siano effettivamente eseguite le operazioni.

- **Scalabilità**

I sistemi distribuiti sono progettati per crescere nel tempo, sia in termini di numero di utenti sia di quantità di dati o nodi coinvolti. Una buona scalabilità consente di aggiungere nuove risorse (server, servizi, istanze) senza dover riprogettare l'intero sistema e senza degradare significativamente le prestazioni.



Le quattro caratteristiche principali dei sistemi distribuiti possono essere sintetizzate e spiegate come segue:

- **Tolleranza ai guasti**

Poiché i componenti di un sistema distribuito possono guastarsi indipendentemente, il sistema deve essere in grado di continuare a funzionare anche in presenza di errori parziali. Questo risultato si ottiene tramite meccanismi come la replica dei dati, il failover e il rilevamento automatico dei guasti.

- **Concorrenza**

In un sistema distribuito più processi o utenti possono accedere simultaneamente alle stesse risorse. La gestione della concorrenza è fondamentale per garantire correttezza e coerenza dei dati, evitando problemi come condizioni di race, inconsistenze o accessi non sincronizzati.



I principali sistemi distribuiti rappresentano diverse modalità con cui l'elaborazione e la gestione delle risorse vengono suddivise su più nodi cooperanti, ciascuna pensata per rispondere a specifiche esigenze di scalabilità, affidabilità e prestazioni.

Nel tempo, l'evoluzione delle reti e delle applicazioni ha portato alla nascita di modelli architetturali differenti, oggi ampiamente utilizzati in ambito enterprise, web e cloud. Di seguito sono riportate le tipologie più rilevanti, accompagnate da una breve descrizione.

Sistemi client-server

- Basati su una separazione netta dei ruoli, in cui i client richiedono servizi e i server li forniscono. Sono il modello più tradizionale e costituiscono la base di molte applicazioni web e gestionali.

Sistemi a più livelli (multi-tier)

- Estendono il modello client-server suddividendo l'applicazione in livelli logici (presentazione, logica applicativa, dati), migliorando manutenibilità, sicurezza e scalabilità.



- **Sistemi peer-to-peer (P2P)**

Tutti i nodi hanno pari ruolo e possono agire sia da client sia da server.
Questo approccio elimina il punto centrale di controllo e favorisce la scalabilità e la resilienza, ma rende più complessa la gestione globale.

- **Sistemi basati su microservizi**

L'applicazione è composta da servizi indipendenti, distribuiti e comunicanti tramite API. Ogni microservizio può essere sviluppato, distribuito e scalato autonomamente.

- **Sistemi distribuiti per la gestione dei dati**

Comprendono database distribuiti e sistemi di storage che replicano e partizionano i dati su più nodi per garantire disponibilità, prestazioni e tolleranza ai guasti.

- **Sistemi cloud e cluster**

Utilizzano grandi insiemi di macchine coordinate per offrire risorse di calcolo, storage e servizi on demand, rappresentando oggi uno degli esempi più diffusi di sistemi distribuiti su larga scala.



I punti di contatto tra sistemi distribuiti e Big Data sono numerosi, poiché le tecnologie Big Data si basano quasi interamente su principi e architetture tipiche dei sistemi distribuiti. In particolare:

- **Distribuzione dei dati**

I Big Data implicano volumi di dati tali da non poter essere gestiti da un singolo nodo. I dati vengono quindi partizionati e distribuiti su più macchine, secondo logiche di shard e replica tipiche dei sistemi distribuiti.

- **Scalabilità orizzontale**

Sia i sistemi distribuiti sia le piattaforme Big Data puntano ad aumentare le prestazioni aggiungendo nodi al sistema, anziché potenziarne uno solo. Questo approccio consente di gestire la crescita continua dei dati e delle richieste.



I punti di contatto tra sistemi distribuiti e Big Data sono numerosi, poiché le tecnologie Big Data si basano quasi interamente su principi e architetture tipiche dei sistemi distribuiti. In particolare:

- Tolleranza ai guasti

Nei contesti Big Data il guasto di singoli nodi è considerato normale. I sistemi distribuiti forniscono i meccanismi (replica, ricalcolo, riassegnazione dei task) che permettono di continuare l'elaborazione senza perdita di dati o interruzioni del servizio.

- Elaborazione parallela

L'analisi dei Big Data avviene tramite l'esecuzione parallela di operazioni su insiemi di dati distribuiti. Questo modello sfrutta il calcolo distribuito per ridurre drasticamente i tempi di elaborazione.



I punti di contatto tra sistemi distribuiti e Big Data sono numerosi, poiché le tecnologie Big Data si basano quasi interamente su principi e architetture tipiche dei sistemi distribuiti. In particolare:

- **Consistenza e coordinamento**

La gestione dei Big Data richiede un equilibrio tra consistenza, disponibilità e prestazioni. I sistemi distribuiti forniscono i modelli teorici e pratici per coordinare nodi indipendenti e garantire risultati affidabili.

- **Astrazione dell'infrastruttura**

In entrambi i casi l'obiettivo è nascondere la complessità dell'hardware sottostante, permettendo a sviluppatori e analisti di concentrarsi sui dati e sulla logica applicativa.



I sistemi distribuiti e i Big Data diventano realmente operativi grazie a strumenti che gestiscono distribuzione dei dati, coordinamento dei nodi, calcolo parallelo e scalabilità automatica.

Questi tool astraggono la complessità dell'infrastruttura e permettono di costruire pipeline affidabili, tolleranti ai guasti e in grado di operare su grandi volumi di dati in modo efficiente.

- **Apache Hadoop**

Framework storico per il Big Data basato su storage distribuito (HDFS) ed elaborazione batch. È un esempio classico di sistema distribuito fault-tolerant orientato ai grandi volumi di dati.

- **Apache Spark**

Motore di calcolo distribuito in-memory, progettato per analisi veloci, machine learning e data processing su cluster. Riduce la latenza rispetto ai modelli batch tradizionali.



- **Apache Kafka**

Sistema di messaggistica e streaming distribuito, fondamentale per l'integrazione tra servizi, l'elaborazione di eventi in tempo reale e le architetture data-driven.

- **Apache Flink**

Framework per elaborazioni distribuite in streaming e near real-time, molto usato quando è richiesta bassa latenza e gestione precisa dello stato.



Tool per storage e database distribuiti

- **Apache Cassandra**

Database NoSQL progettato nativamente come sistema distribuito, altamente scalabile e tollerante ai guasti, ideale per grandi volumi di dati e accessi concorrenti.

- **MongoDB**

Database documentale con supporto a replica e sharding, utilizzato in contesti distribuiti per applicazioni scalabili e orientate ai dati.

- **Amazon S3**

Storage a oggetti distribuito, spesso utilizzato come data lake per sistemi Big Data e pipeline analitiche su larga scala.



Tool per orchestrazione, coordinamento e infrastruttura

- **Kubernetes**

Piattaforma di orchestrazione per sistemi distribuiti basati su container.
Gestisce scalabilità, resilienza e deploy automatici di servizi e job Big Data.

- **Docker**

Consente di impacchettare applicazioni e servizi in container portabili,
facilitando la distribuzione coerente su più nodi.

- **Apache ZooKeeper**

Servizio di coordinamento distribuito utilizzato per gestione della configurazione, elezione dei leader e sincronizzazione tra nodi.



Tool cloud per sistemi distribuiti e Big Data

- **Amazon Web Services, Microsoft Azure, Google Cloud Platform**

Offrono servizi gestiti per Big Data e sistemi distribuiti (cluster, data lake, streaming, ML), riducendo drasticamente la complessità infrastrutturale.



I sistemi distribuiti, pur offrendo numerosi vantaggi, presentano anche limiti strutturali e concettuali che ne aumentano la complessità progettuale e gestionale.

Comprendere questi limiti è fondamentale per valutarne l'adozione in modo consapevole.

- Complessità di progettazione e sviluppo

La realizzazione di un sistema distribuito richiede la gestione di comunicazioni di rete, sincronizzazione, concorrenza e gestione degli errori parziali. Questo rende il design più complesso rispetto ai sistemi centralizzati.

- Problemi di consistenza dei dati

Mantenere i dati coerenti tra più nodi distribuiti è difficile, soprattutto in presenza di repliche e aggiornamenti concorrenti. Spesso è necessario accettare compromessi tra consistenza, disponibilità e prestazioni.



- **Latenza e dipendenza dalla rete**

Le comunicazioni tra nodi avvengono tramite rete, introducendo ritardi, congestioni e possibili interruzioni. Le prestazioni complessive dipendono fortemente dalla qualità e dall'affidabilità della rete.

- **Gestione dei guasti parziali**

In un sistema distribuito i guasti non sono totali ma parziali: alcuni nodi possono fallire mentre altri continuano a funzionare. Rilevare e gestire correttamente questi scenari è complesso e non sempre immediato.

- **Difficoltà di debugging e monitoraggio**

Individuare errori, colli di bottiglia o comportamenti anomali è più difficile rispetto a un sistema monolitico, poiché il problema può emergere solo dall'interazione tra più componenti distribuiti.





Buon Davante a tutti