UNIVERSITÀ POLITECNICA DELLE MARCHE

Facoltà d'Ingegneria

Dipartimento Ingegneria dell'Informazione

Corso di Laurea magistrale in Ingegneria Elettronica

Valutazione delle prestazioni di una infrastruttura di rete Blockchain privata per industria 4.0

Performance Assessment of a Private Blockchain Network Infrastructure for Industry 4.0

Relatore

Chiar.mo Prof. Marco Baldi

Laureando

Davide Cesario Bevilacqua

Correlatori

Dr. Paolo Santini

Dr.ssa Giulia Rafaiani

A. A. 2020/2021

INDICE

1. INTRODUZIONE

L’avvento della digitalizzazione, la propensione all’automatizzazione e all’Internet of Things (IoT) ha generato una nuova sfida tra i maggiori competitors dell’industria globale, cioè l’applicazione di un ambiente “smart” all’interno di un sistema di produzione, per migliorarne efficienza, qualità, precisione e produttività; oggi questo tema è definito *Industria 4.0*.

Nella definizione stessa di “catena di produzione” esiste uno scompenso [1] tra costo, flessibilità, velocità e qualità. Sebbene siano tutte proprietà importanti, un'azienda non riesce a garantire il massimo su tutte, arrivando a scegliere su quali puntare. Le tecnologie alla base di Industria 4.0 possono migliorare una o più priorità arrivando anche a compensare in equilibrio lo scompenso tra di esse. L’obiettivo, quindi, è soddisfare in modo costante ed efficiente la domanda dei clienti, in costante crescita, sfruttando l'applicazione di nuovi strumenti tecnologici e solide strutture organizzative che lavorano in sinergia per sviluppare servizi o prodotti.

Nonostante ciò, alcune delle difficoltà più comuni che si trovano affrontando problemi riguardanti Industria 4.0 sono l’enorme numero di dispositivi interconnessi tra di loro, il decentramento e le connessioni instabili ed imprevedibili, da cui si possono evidenziare le sfide più ardue che si devono affrontare quando si entra nel tema di Industria 4.0 [2]:

1. l’eterogeneità, ovvero la comunicazione e l’interoperabilità tra dispositivi diversi interconnessi tra loro,
2. la complessità delle reti, l’uso di protocolli di rete che coesistono nelle applicazioni IoT poiché non c’è uno standard da seguire,
3. la scarsa interoperabilità, ovvero la scarsa collaborazione tra i dispositivi in uso,
4. i limiti delle risorse dei dispositivi IoT; risorse come memoria, capacità di elaborazione, capacità di archiviazione, alimentazione e larghezza di banda,
5. le vulnerabilità della privacy, ovvero l’uso appropriato dei dati raccolti, che non devono essere divulgati senza autorizzazione del proprietario,

6. le vulnerabilità della sicurezza, aspetto molto importante per qualsiasi applicazione industriale; l'eterogeneità e il decentramento rendono questo aspetto il più difficile da affrontare,
7. l'organizzazione di grandi quantità di dati, in termini di comunicazione/trasmissione e archiviazione per la mole di dati che vengono generati.

Tutto ciò, come precedentemente scritto, è dato dalla variegata disponibilità di molteplici e diversi strumenti utilizzabili. Nella Fig. 1, vengono mostrati le principali categorie di dispositivi che si potrebbero trovare trattando di Industria 4.0 [2].

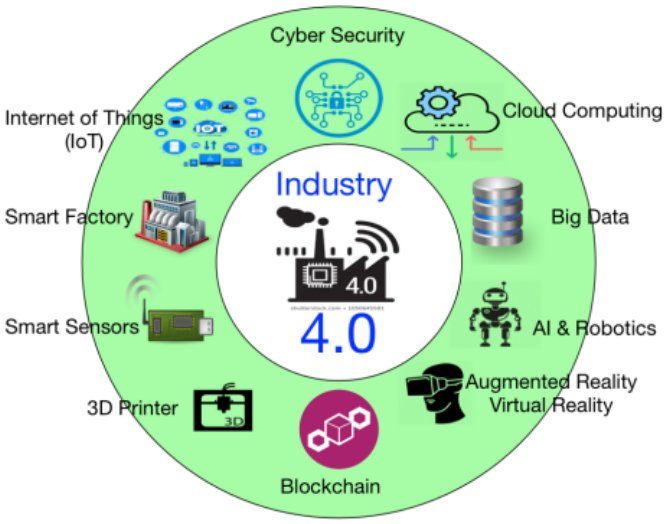

Figura 1: Industria 4.0: Principali componenti [2]

La trasformazione digitale è un processo di sviluppo che porta efficienza e competitività. Con l'aggiunta delle tecnologie blockchain, l'attuale rivoluzione digitale porterà innovazione alle tecnologie per Industria 4.0, infatti ci sono alcuni vantaggi nell'uso della blockchain per affrontare alcune problematiche [3].

Una delle problematiche è l'identificazione certificata. Una blockchain potrebbe offrire l'equivalente digitale di un documento di riconoscimento che può essere utilizzato per identificare non solo le persone ma anche enti diversi come le organizzazioni, consentendo di riconoscere ed autenticare tutte le entità coinvolte in qualsiasi attività industriale su una rete pubblica. Queste identità digitali possono essere emesse da un'organizzazione

governativa come se fosse il rilascio di patenti di guida, passaporti, registrazioni di società e titoli di proprietà.

Un’altra problematica è la sicurezza distribuita. Una delle principali caratteristiche di successo della blockchain è la capacità di proteggere i dati e le transazioni scritte nei registri condivisi utilizzando un approccio compartimentato e distribuito. Questa protezione avviene tramite serie concatenate di crittografie, che rendono impossibile alterare qualsiasi record che sia già stato aggiunto alla catena, infatti ogni nuova transazione, dopo essere stata convalidata dagli enti partecipanti, è collegata alla catena delle transazioni precedenti e nessun record può essere modificato in alcun modo. Inoltre, basandosi sulle identità digitali verificate e sul fatto che ogni transazione è registrata con un pieno accordo tra tutti i soggetti coinvolti, diventa praticamente impossibile per uno di questi soggetti negare successivamente di essere coinvolto o concordato. L'approccio utilizzato consente una migliore protezione delle transazioni e minori rischi di esposizione in caso di violazioni della sicurezza.

Le funzionalità di registrazione, convalida e sicurezza della blockchain, oltre al supporto dell'identità digitale, consentono i cosiddetti smart contract che conducono operazioni attendibili su una rete pubblica senza terzi. Uno smart contract è tracciabile, sicuro e inalterabile. Uno degli usi dello smart contract è l'automazione dei processi di accordo tra le aziende e i loro partner e tra le aziende e i loro client. Ciò ridurrà notevolmente i costi amministrativi e creerà un modello efficiente per avviare, negoziare e concludere contratti senza la necessità di operazioni terze o pesanti documentazioni.

Un'altra area in cui le funzionalità blockchain possono avere un impatto positivo sulla catena di produzione è la capacità di facilitare la micro-misurazione. La capacità di registrare in modo sicuro eventi e attività senza la necessità di conferme di terze parti e garanzie esterne aumenterà la quantità di dati e attività registrate e consentendo alle organizzazioni di creare registri dettagliati delle loro attività. Questi possono essere facilmente analizzati per fornire misurazioni e controlli di qualità a qualsiasi livello di dettaglio.

Di fronte a queste situazioni, la blockchain è un ottimo strumento che offre grandi potenzialità, ottenendo di conseguenza efficienza, sicurezza, robustezza e decentralizzazione.

Lo scopo di questo lavoro è quello di implementare una blockchain privata, avviarla su hardware diversi e capire il modo per gestirla, inoltre, una volta raggiunto il regime di

funzionamento, trovare metodi per testarla e monitorare sia l'andamento della catena stessa sia come quest'ultima vada a usufruire delle risorse dell'hardware su cui è stata avviata. Verranno, inoltre, illustrate le particolarità della blockchain privata scelta, descritti gli strumenti usati e spiegati quali test sono stati effettuati. In conclusione verranno illustrati e, infine, confrontati i risultati ottenuti in modo da stimarne il comportamento a regime su hardware differenti tra di loro.

2. BLOCKCHAIN

La blockchain è una tecnologia nota principalmente per il suo utilizzo nella gestione delle criptovalute, come Bitcoin. La blockchain può essere definita approssimativamente come un registro immutabile, decentralizzato, affidabile e condiviso su reti distribuite (esempio: peer-to-peer (P2P)). In sostanza, la blockchain è una struttura di dati distribuita, ed è etichettata come "Distributed Ledger". Questa tecnologia ha un grande potenziale per essere adottata in altri scenari in cui avvengono scambi di dati.

L'idea chiave alla base della tecnologia blockchain è il decentramento, ciò significa che non è richiesta nessuna autorità centrale per controllare o gestire i nodi partecipanti; tutti i nodi partecipanti, chiamati *peers,* in una rete abilitata alla blockchain mantengono copie identiche del registro.

Ogni nodo ha la possibilità di verificare il comportamento di altri nodi interni alla rete, aggiungere blocchi alla catena nonché la capacità di creare, autenticare e convalidare nuove transazioni, che verranno registrate nei blocchi scritti nella catena. Questa architettura decentralizzata garantisce operazioni robuste, sicure ed efficienti.

Una blockchain può essere definita in due modi:

- pubblica: chiunque può entrare a far della blockchain come nodo, e quindi ottenere i permessi di scrittura e lettura dei blocchi nella catena,
- privata: l'attività di scrivere blocchi nella catena è limitata, solo chi ha le autorizzazioni può farlo, mentre l'attività di lettura può essere anche essere pubblica.

Un registro blockchain cresce continuamente se vengono eseguite e aggiunte transazioni. Quando un nuovo blocco viene generato da un nodo partecipante, a seconda del protocollo di consenso, deve passare attraverso il processo di convalida da parte di tutti gli altri nodi.

Un blocco è composto da due parti principali:

- header: dove sono presenti tutti i campi di gestione del blocco,
- body: dove sono presenti i dati delle transazioni.

Il protocollo di consenso è un insieme specifico di regole che permettono ai nodi della rete di assicurare che un blocco venga convalidato nella catena. Ciò deve essere reso possibile anche qualora un nodo potrebbe andare offline o fosse inaffidabile in un qualsiasi momento.

Esistono tre tipi di protocolli di consenso:

- Proof of Work (PoW): è il metodo convenzionale per creare nuovi blocchi dopo aver completato le transazioni. Il funzionamento è che un nodo verifica un blocco di transazione risolvendo un problema matematico basato su numeri primi; una volta confermato, viene convalidata la transazione e viene creato un nuovo blocco assegnando una ricompensa al nodo che completa la transazione.

- Proof of Stake (PoS): un nodo che desidera estrarre o convalidare una transazione sulla blockchain può farlo a seconda di quanti blocchi ha già. Maggiore è il numero di blocchi che il nodo ha nella blockchain, maggiore è la potenza di convalida che gli viene data sulla blockchain.

- Proof of Authority (PoA): è un metodo di consenso in cui a un certo numero di attori blockchain all'interno dell'ecosistema viene dato il potere di convalidare le transazioni e in definitiva decidere se nuovi blocchi verranno aggiunti o meno alla blockchain; è un metodo non adatto per blockchain pubbliche in quanto esiste essenzialmente un monopolio con pochi attori che possono confermare le transazioni.

Una volta che il blocco proposto viene convalidato, viene automaticamente aggiunto alla fine della blockchain tramite un riferimento che punta al blocco immediatamente precedente. Il primo blocco di una blockchain è chiamato blocco di genesi e contiene le regole con cui la blockchain viene mantenuta, come, ad esempio, quale protocollo di consenso viene utilizzato [4].

La Fig. 2 mostra l'evoluzione delle reti di computer da sistemi decentralizzati a sistemi decentralizzati e distribuiti.

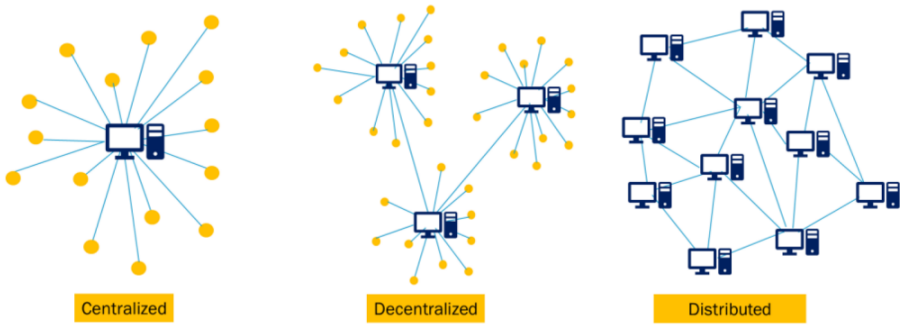

Figura 2: Tre stadi di evoluzione di una rete. fonte:Daxx.com [3]

3. STRUMENTI USATI E CREAZIONE DELLA RETE DI TEST

In questo capitolo verranno trattati gli strumenti usati, verrà spiegata l'infrastruttura blockchain privata scelta, con le sue particolarità e i suoi protocolli di consenso, poi verranno descritti i tool di monitoraggio, con la loro architettura e le loro potenzialità, ed infine verrà descritto il codice javascript per il benchmark generale.

L'obiettivo dello studio è valutare le prestazioni di una blockchain Besu, soprattutto come reagisce l'hardware, specificamente una macchina virtuale e un server dedicato, su cui è stata implementata la rete, utilizzando con protocollo di consenso PoA, monitorando le risorse principali di un nodo validatore.

3.1 HYPERLEDGER BESU

La scelta dell'infrastruttura Blockchain da testare è ricaduta su Hyperledger Besu, per la sua modularità, semplicità di configurazione e possibilità di creare reti private.

Molte aziende sono interessate alle potenzialità delle blockchain private, soprattutto sulla possibilità della creazione di reti autorizzate per le loro applicazioni [5] [6]. L'obiettivo di Besu è quello di diventare una piattaforma open-source duttile per lo sviluppo, facile da aggiornare e da implementare.

Hyperledger Besu è un client Ethereum open source sviluppato con licenza Apache 2.0 e scritto in Java. Può essere eseguito sulla rete pubblica di Ethereum o su reti private, nonché su reti di test come Rinkeby, Ropsten e Görli. Besu permette l'implementazione di diversi algoritmi di consenso, tra cui Proof of Work (PoW) e Proof of Authority (PoA) e dispone di schemi completi per l'autorizzazione, progettati specificamente per l'utilizzo in un caso di rete Pubblica-Permissionata.

Nello specifico, essendo Besu un client Ethereum (il software, quindi, implementa il protocollo Ethereum), possiede:

- un ambiente di esecuzione per l'elaborazione delle transazioni nella blockchain di Ethereum;
- archiviazione per dati persistenti relativi all'esecuzione della transazione;

- rete peer-to-peer (P2P) per la comunicazione con gli altri nodi Ethereum sulla rete per sincronizzare lo status della rete stessa;
- Application Programming Interface (API) utilizzabili dagli sviluppatori di applicazioni per interagire con la blockchain.

Le caratteristiche di Hyperledger Besu sono consultabili direttamente dal sito ufficiale [7].

3.2 ALGORITMI DI CONSENSO IN BESU

Besu propone due algoritmi di consenso, suddivisi in tre diversi meccanismi:

- Ethash (PoW),
- clique (PoA),
- IBFT 2.0. (PoA)

Nel continuo di questo capitolo si andrà a descrivere, in termini generali, il funzionamento di ognuno di questi algoritmi.

ETHASH

Ethash è l'algoritmo PoW progettato per Ethereum 1.0.

L'algoritmo [8] per ogni blocco calcola un seed hash basato sugli header dei vari blocchi scritti fino a quel punto. Dal seed hash, si può calcolare una pseudorandom cache di 16 MB, da cui si può generare un dataset da 1 GB. Tale dataset deve avere una proprietà, ovvero, ogni elemento all'interno di esso deve dipendere solo da un piccolo numero di elementi dalla cache. Il mining comporta l'estrazione di sezioni casuali del dataset e del loro hashing. La verifica può essere eseguita con poca memoria utilizzando la cache per rigenerare i pezzi specifici del dataset di cui si ha bisogno.

CLIQUE

Clique è uno degli algoritmi PoA, progettato per seguire i meccanismi della blockchain Ethereum [9].

Essendo un PoA, Clique è un meccanismo di consenso in cui vengono definiti *authorities* i nodi che convalidano i blocchi all'interno di un certo periodo. Si ottiene così un sistema in cui, dividendo il tempo in *steps*, ogni *step* ha uno ed un solo *leader authority* che convalida il blocco. Ogni *authority* potrà essere leader e convalidare un blocco solo ogni $\left(\frac{N}{2}+1\right)$ blocchi, e dopo solo $N-\left(\frac{N}{2}+1\right)$ step.

IBFT 2.0

Instanbul Bizantine-Fault-Tolerant IBFT è un protocollo PoA che riesce ad adattarsi a quella classe di fallimenti derivata dal "*Problema dei Generali Bizantini"*. Ciò significa che è in grado di continuare ad operare anche se alcuni nodi falliscono o agiscono in modo disonesto.

Per convalidare un blocco, il consenso della rete deve raggiungere il 66% dei validatori attivi. Ad esempio, se la rete ha 5 validatori, con l'accordo di soli 3 nodi, il blocco viene convalidato e si aggiunge alla blockchain. La Fig. 3 mostra in modo esemplificativo come un nodo propositore può raggiungere il consenso tramite due distinti set di validatori.

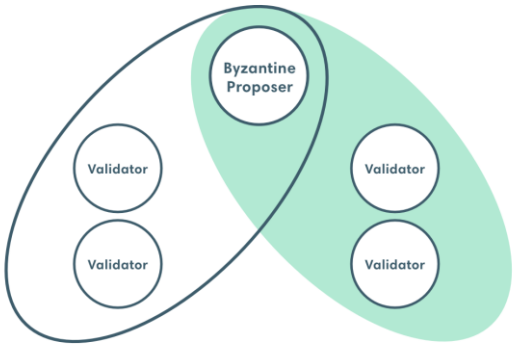

Figura 3: Nodo Bizantino. (fonte: consensys.net [10])

Besu implementa la IBFT 2.0 che consente alla rete di sfruttare le capacità degli smart contract di Ethereum, con alcune proprietà aggiuntive riguardanti la robustezza della rete, come il fatto che i validatori per convalidare il blocco devono essere almeno 4 [10].

3.3 TOOLS DI MONITORAGGIO

Prima di avviare la rete, dopo un'attenta ricerca, sono stati trovati e configurati dei tools per monitorare statisticamente sia le risorse hardware che la rete stessa.

PROMETHEUS

Prometheus è un toolkit di monitoraggio open source che permette di estrarre dati, come l'utilizzo della CPU, la RAM occupata o i dati immagazzinati nel file system, in tempo reale. Questo strumento ha permesso il monitoraggio delle risorse della macchina su cui la rete è stata avviata e il monitoraggio delle prestazioni della rete Blockchain stessa [11].

L'ecosistema Prometheus è costituito da più componenti, molti dei quali sono opzionali:

- il server principale di Prometheus che raccoglie e memorizza i dati delle serie temporali;
- librerie client per la strumentazione del codice dell'applicazione;
- esportatori speciali per servizi come HAProxy, StatsD, Graphite, ecc.

La maggior parte dei componenti di Prometheus sono scritti utilizzando il linguaggio Go, il che li rende facili da compilare e da distribuire come binari statici. Il diagramma in Fig. 4 illustra l'architettura di Prometheus e alcuni dei componenti del suo ecosistema.

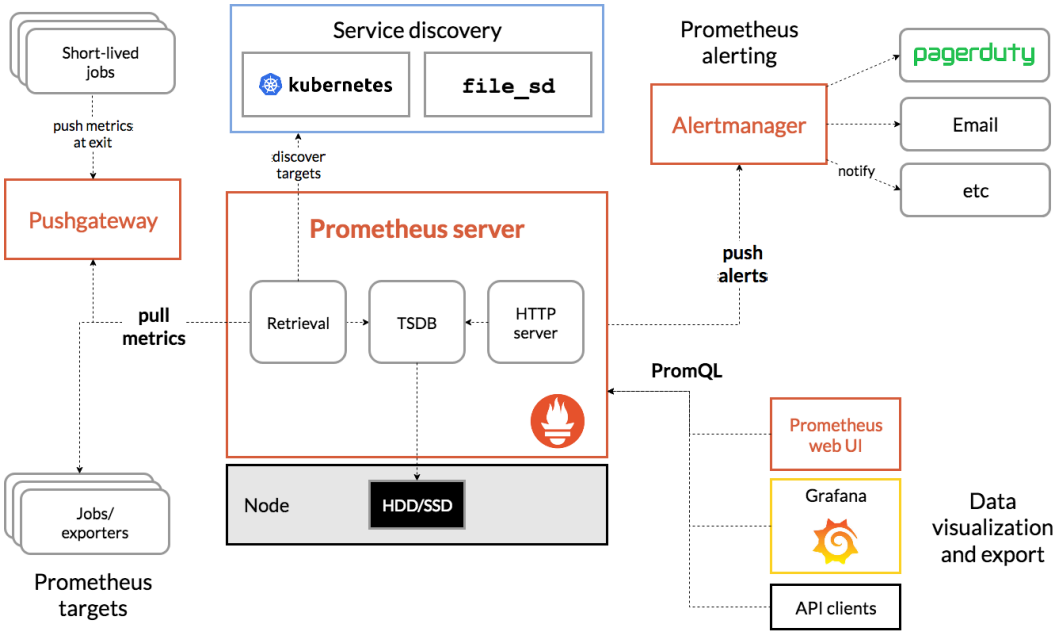

Figura 4: Architettura Prometheus [11]

Prometheus ha due modi per l'estrazione dei dati, o direttamente dai client in esecuzione (tipo Besu) o tramite un gateway push intermedio, memorizza localmente tutti i dati raccolti e su di essi può eseguire delle operazioni in modo da arrivare ad aggregare i dati stessi tra di loro. In più permette ad altre app di terze parti basate su Client API (tipo Grafana) di utilizzare, visualizzare e quindi eseguire operazioni sui dati raccolti. Prometheus si adatta bene sia al monitoraggio della macchina stessa che al monitoraggio di architetture dynamic service-oriented,

NODE EXPORTER

Node Exporter è un plugin di Prometheus per metriche a livello di server e di sistema operativo. Questo strumento aiuta nel misurare le varie risorse del server come RAM, spazio su disco e utilizzo della CPU. Questo plugin è risultato una buona soluzione per raccogliere tutte le statistiche relative al monitoraggio delle risorse del server Linux.

GRAFANA

Grafana è un software di visualizzazione e analisi open source. Consente di interrogare, visualizzare ed esplorare i dati raccolti, indipendentemente da dove sono archiviati. In parole povere, fornisce gli strumenti per estrarre i dati del database delle serie temporali.

Nel caso in esame, Grafana è utile perché si interfaccia con il Server Prometheus e permette di estrarre i dati raccolti ed analizzarli successivamente, in modo pratico e veloce.

3.4 PREPARAZIONE DEI TEST

Prima di tutto si è dovuto decidere le impostazioni del benchmark, cioè i test da effettuare.

Il test prestabilito consistere nell'inviare diversi cicli di transazioni, per un tempo prestabilito equivalente a 30 minuti, modificando numero di transazioni (Tx) inviate e nel mentre monitorare le risorse del nodo. Più precisamente, è stato deciso di generare tre cicli di transazioni:

- 1 transazione a blocco,
- 10 transazioni a blocco,
- 100 transazioni a blocco.

Le risorse che si è scelto di monitorare sono:

- CPU,
- RAM,
- Hard Disk,
- Memoria Java Virtual Machine.

Besu si avvia su una Java Virtual Machine (JVM). La JVM è una macchina informatica astratta che consente a un computer di eseguire un programma Java; per questo, è interessante capire come la blockchain gestisca la memoria della Virtual Machine (VM) su cui viene eseguita. La memoria JVM è divisa in due parti: memoria heap e memoria non heap.

MEMORIA HEAP

La memoria heap è l'area dei dati di runtime su cui viene allocata la memoria per tutte le istanze e gli array della classe Java. L'heap viene creato all'avvio di Java Virtual Machine e può aumentare o diminuire di dimensioni durante l'esecuzione dell'applicazione. Praticamente è la parte volatile della JVM.

MEMORIA NON-HEAP

La Java Virtual Machine dispone di una memoria diversa dall'heap, denominata memoria non heap. Viene creato all'avvio della JVM e memorizza strutture per classe come pool di

costanti di runtime, dati di campi, metodi, il codice per metodi e costruttori, nonché stringhe interne. La dimensione massima predefinita della memoria non heap è 64 MB. La non heap è la parte statica della JVM.

Si è cercato di capire, successivamente, quali sono i requisiti di sistema per la blockchain o da quali fattori dipende l'avvio di una rete, questi ultimi sono:

- dimensioni della rete Blockchain,
- numero di transazioni inviate alla rete,
- limite massimo di gas utilizzabile per blocco,
- numero e complessità delle query JSON-RPC gestite dal nodo.

La capacità minima di memoria JVM per le reti private è di 4 GB. I requisiti di memoria JVM, però sono più elevati durante la sincronizzazione, ma vengono ridotti dopo che il nodo si sincronizza con la catena.

Quindi, si è deciso inizialmente di avviare Hyperledger Besu su una macchina Virtuale (Sistema Operativo: Lubuntu 20.10 – 64bit) con le seguenti caratteristiche:

- CPU: Single Core - Intel Core i5 – 9600K;
- RAM: 4GB;
- HDD: SDD 50GB.

Inoltre, per i test si è deciso di avviare una rete permissionata, tramite Proof of Authority, con il meccanismo di consenso Clique ad un solo nodo.

Dopo aver concluso i test su macchina virutale, si è deciso di ripetere gli stessi test su una macchina server, con le seguenti caratteristiche:

- PC: Intel i9 10900X a 10 core;
- RAM: 32 GB;
- SSD: 1 TB.

3.5 FILE DI GENESI E ISTRUZIONI DI AVVIO

Per avviare la rete è indispensabile il file di genesi. Il file di genesi definisce il primo blocco della catena che verrà avviata; all'interno si trovano, infatti, tutti dati del blocco, nonché tutte le regole per la blockchain stessa. Se un nodo volesse tentare di far parte della rete, per avviarsi e collegarsi dovrà usare il file di genesi come punto di partenza. Tra le impostazioni su tutte più interessanti si trovano:

- ***clique, ibft2.0***: definizione del protocollo di consenso da usare.
- ***epochlength*:** indica il numero di blocchi a cui azzerare, eventualmente, i voti in sospeso nella rete.
- ***blockperiodseconds***: impostazione che indica quanti secondi servono per validare un blocco.
- ***extraData:*** contiene l'indirizzo del primo nodo da cui far partire la blockchain.
- ***gasLimit*:** indica il limite massimo di gas utilizzabile per blocco.
- ***mixHash:*** identificatore univoco per il blocco.
- ***alloc***: impostazione per definire gli account iniziali con il proprio saldo.

Il file di genesi utilizzato per avviare la rete in esame è riportato in Appendice A. Il valore ***blockperiodseconds*** permette di modificare il tempo di convalida, questo valore coincide con il tempo di transazioni al secondo, poiché le transazioni saranno inserite nel blocco che poi sarà convalidato.

Dopo aver scritto il file di genesi, è stato avviato il nodo.

Le istruzioni utilizzate sono riportate in Appendice B. Alcune istruzioni sono state scelte per permettere al nodo di interfacciarsi con client esterni, alcune invece sono state scelte per essere impostate in modo da facilitare operazioni nella rete stessa. Di seguito sono riportati alcuni dei comandi più importanti:

- *genesis-file*: imposta il file di genesi;
- *rpc-http-api*: imposta le API con cui comunicare con il nodo;
- *rpc-http-max-active-connections:* permette al nodo di avere più connessioni attive;
- *metrics-enabled:* attiva le impostazioni per le metriche;

- *min-gas-price:* imposta il limite di gas minimo per effettuare una transazione.

Il comando *rpc-http-max-active-connections* è stato impostato usando un numero molto alto per permettere al nodo di avere più connessioni attive, e quindi possibilità di inviare molte transazioni contemporaneamente.

Il comando *min-gas-price* è stato impostato a 0, per permettere di inviare le transazioni tra account senza che questi avessero gas (elemento che indica una commissione) sufficiente per effettuarle.

Mentre *metrics-enabled* permette di monitorare il nodo attraverso Prometheus.

È possibile trovare le istruzioni dettagliate per avviare una rete Clique in [12] e per avviare una rete con IBFT in [13].

3.6 CODICE PER BENCHMARK

Per effettuare i test, non esistendo tools certificati di benchmark,sono stati creati due script *javascript*, utilizzando le librerie *web3js* ed avviandoli tramite Visual Studio.

1) Creazione Account

Il primo script (Appendice C) permette di interfacciarsi con la rete generando gli account per effettuare le transazioni. Nel codice si possono vedere tre metodi importanti ripresi dalle librerie web3js:

- *web3('address:port')*: metodo che indica l'indirizzo su cui lo script deve connettersi;
- *web3.eth.accounts.create(String)*: metodo che crea gli account basandosi su una stringa;
- *web3.utils.randomHex(32)*: metodo che genera un numero esadecimale casuale a 32 bit.

Una volta creati gli account, questi ultimi sono scritti in un file *Address.json* per poi essere riutilizzati più avanti.

2) Invio multiplo di transazioni

La particolarità del secondo script (Appendice D) sta nella definizione di una funzione per creare la transazione. Sono presenti due cicli *for*, uno all'interno della funzione *Transaction(Alice, Bob)*, dove Alice e Bob sono account mittente ed account destinatario, che indica quante volte viene effettuata la creazione e l'invio di una transazione tra i due indirizzi, il secondo *for* è fuori e indica quante volte deve effettuare la funzione definita precedentemente. Il primo *for* scandisce quante volte si scrive nella blockchain, il secondo *for* quante transazioni devono essere effettuate. Questo script permette di impostare quanti blocchi devono essere scritti ed inviare contemporaneamente transazioni multiple alla rete. Gli indirizzi vengono estratti dal file *Address.json,* creato con lo script descritto in precedenza.

Tra i metodi più importanti ci sono:

- *web.eth.account.signTransaction*: genera la transazione;
- *web.eth.sendSignedTransaction*: invia la transazione;
- *{nomeTransazione}.rawTransaction:* estrae la raw della transazione;
- *{Transazioneinviata}.transactionHash:* estrae l'hash della transazione avvenuta e registrata nel blocco.

4. FASE DI TEST

Dopo aver avviato la rete con protocollo clique, si passa alla fase di test.

Sono stati effettuati in macchina virtuale due test, con tempo di convalida, che verrà definito Time Signature (TS), a 5 secondi e ad 1 secondo. Successivamente, avendo un server dedicato, è stato effettuato lo stesso test con TS ad 1 secondo.

Il TS va modificato nel file di genesis, modificando il valore a ***blockperiodseconds.*** Per definire invece il numero di transazioni sono stati modificati i due cicli *for* degli script.

Ed infine da questi test sono stati estratti i cicli precedentemente definiti per analizzare la rete nelle sue condizioni a regime.

4.1 PRIMO TEST: TS A 5 SECONDI

Dopo aver impostato il TS a 5 secondi, si sono andati ad avviare il test tramite gli script, impostando i tre cicli. In Fig.5 è mostrato il numero di transazioni scritte nel test, per un certo intervallo temporale, sono evidenziati tutti i 3 cicli decisi precedentemente.

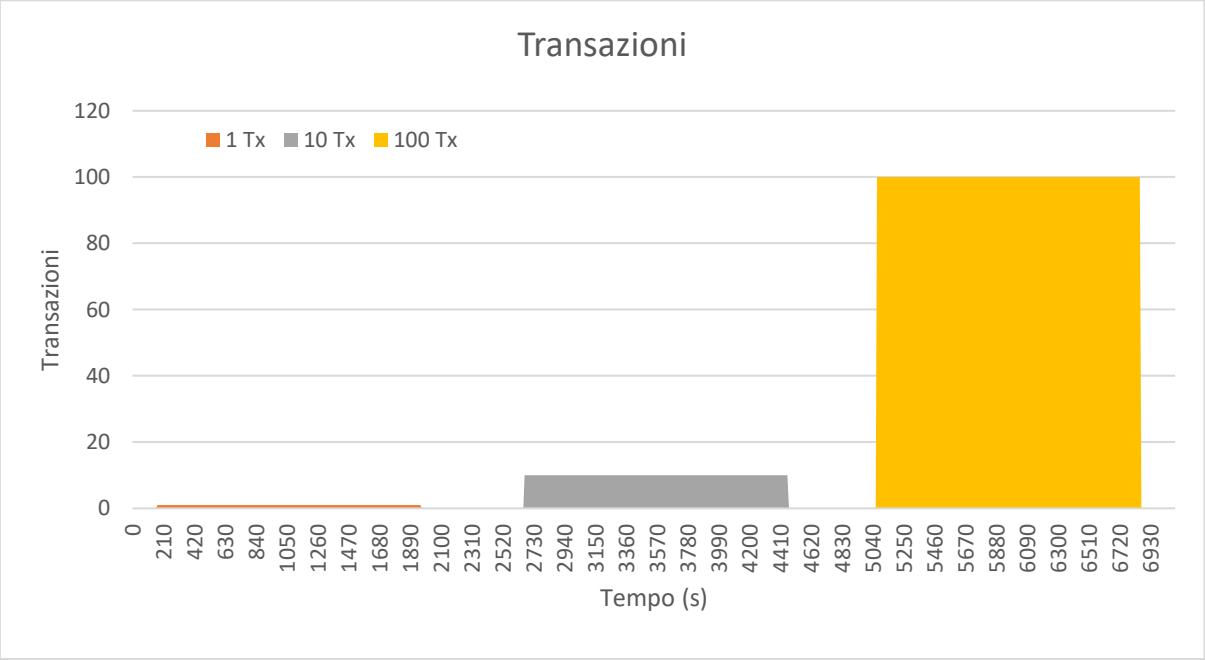

Figura 5: Transazioni per blocco – TS = 5s

La situazione è continua, non c'è volatilità, in tutti e tre i casi c'è una situazione stabile.

Si è passati quindi a monitorare la CPU, come illustrato in Fig. 6.

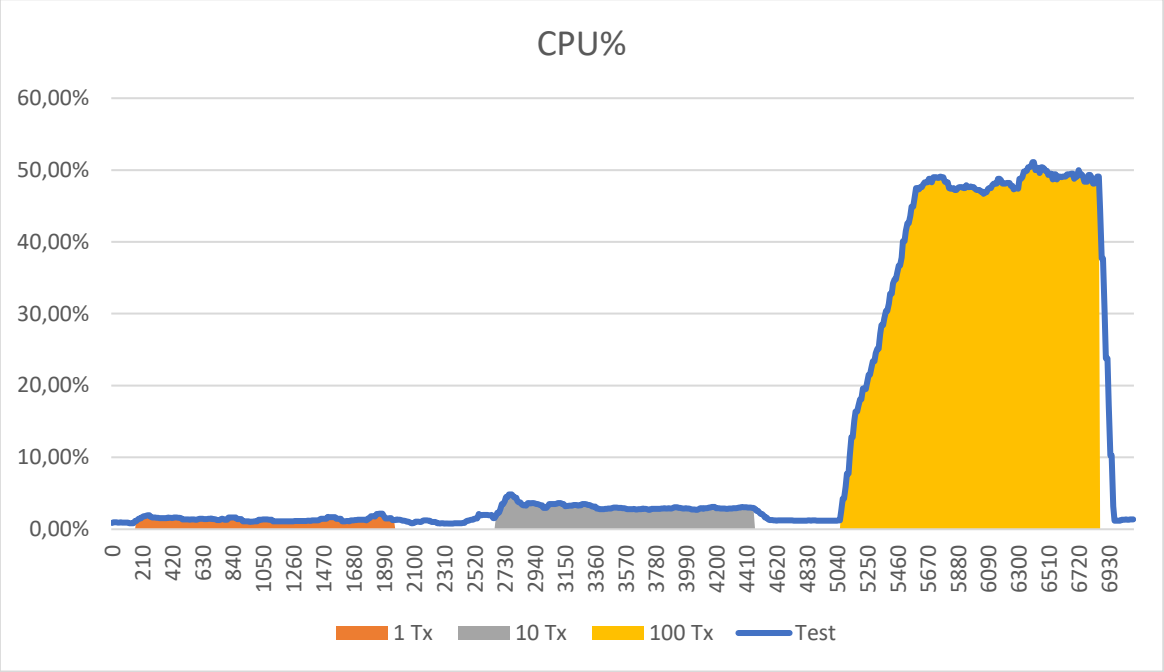

Figura 6: Uso CPU% - TS= 5s

La CPU viene usata maggiormente nel ciclo a 100 Tx. Si nota come a più transazioni c'è una richiesta di CPU maggiore; in Fig.7, c'è il campionamento dei cicli, e si nota che c'è un grande distacco tra il ciclo a 100 Tx con gli altri due.

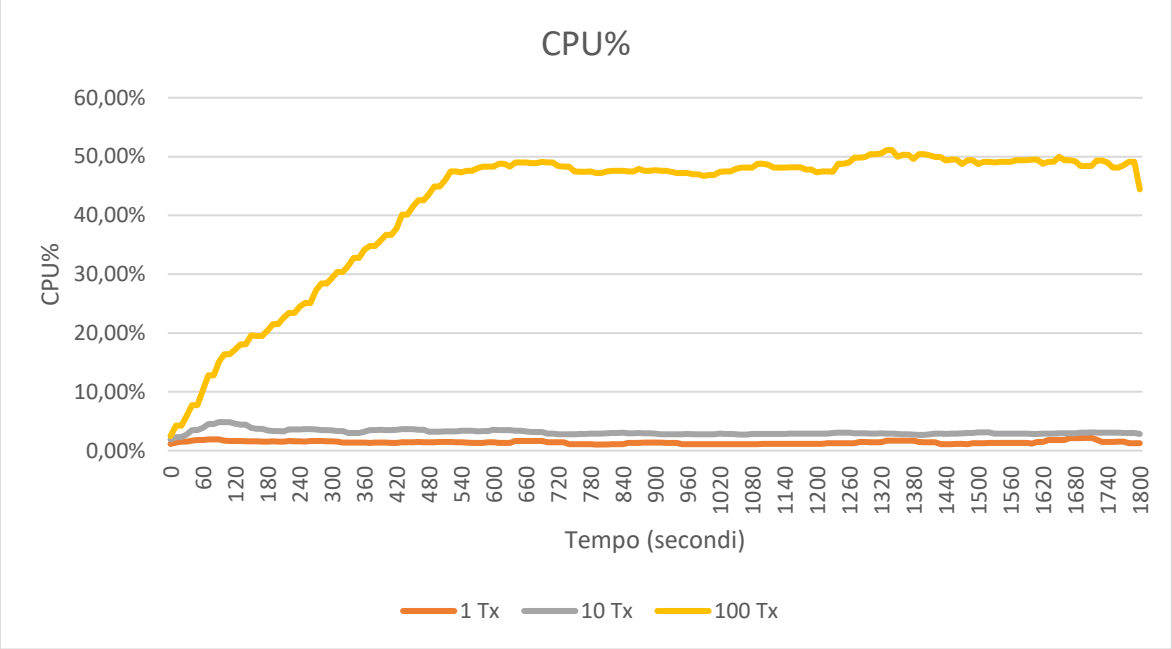

Figura 7: Uso CPU% - Campionamento dei cicli - TS = 5s

Nella Tab. 1 vengono mostrati il massimo, il minimo e la media delle misurazioni dei vari cicli.

Tabella 1: Dati uso CPU% - TS = 5s

	1 Tx	10 Tx	100 Tx
Max	2,17%	4,84%	51,10%
Min	1,03%	1,90%	2,48%
Media	1,41%	3,13%	42,01%

Nello specifico la differenza tra i cicli da 1 Tx e da 10 Tx è minima, mentre confrontandoli con il ciclo a 100 Tx la differenza aumenta in modo netto, infatti passiamo dal 2%-4% al 51% di utilizzo. Quindi a maggior numero di transazioni inviate, c'è un maggior utilizzo della CPU.

Altro dato interessante è capire come Besu pesi sulla RAM. Una rappresentazione dell'occupazione della RAM in funzione del tempo è mostrata in Fig. 8.

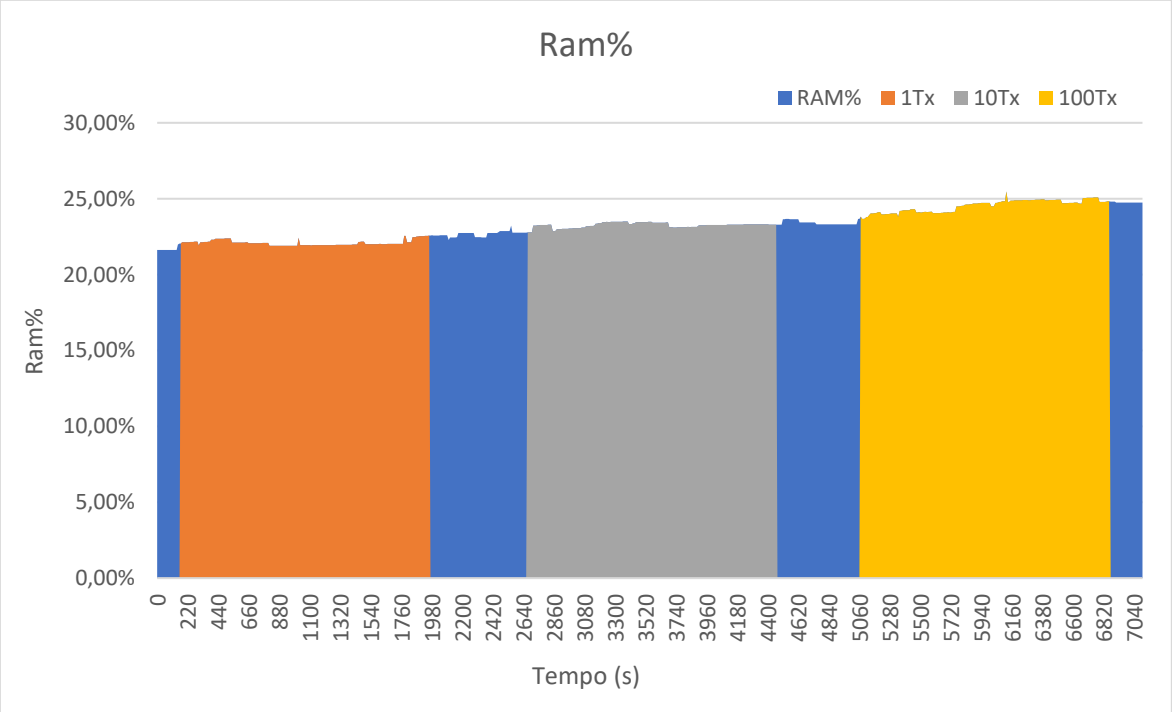

Figura 8: Uso RAM % - TS = 5

La RAM risulta molto stabile e non raggiunge valori altissimi. Analizzando nel dettaglio, la situazione dove viene richiesta più RAM è sempre il caso in cui arrivano 100 Tx a blocco.

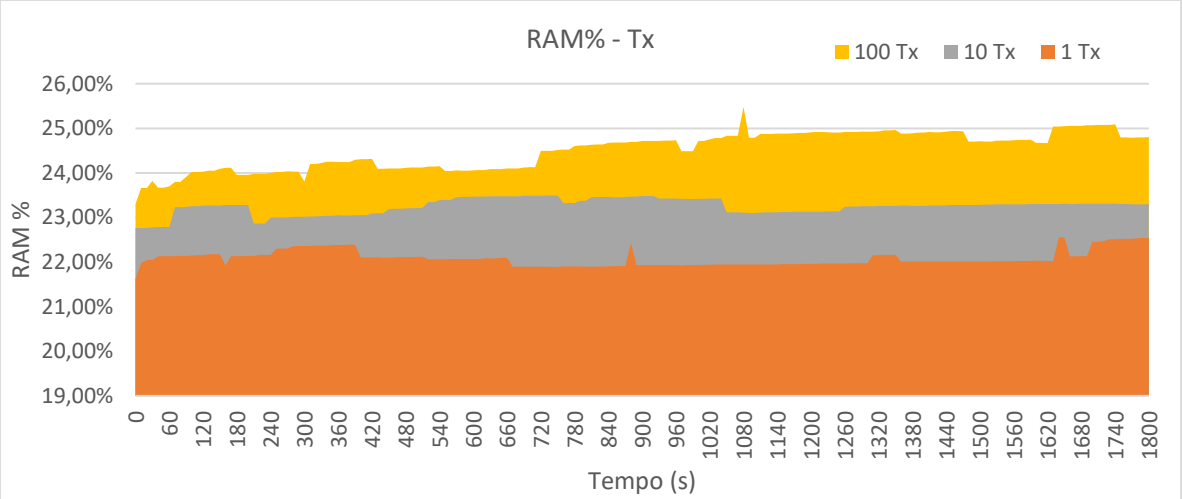

Figura 9: Uso RAM % - Campionamento dei cicli - TS = 5s

Nel periodo di campionamento (Fig. 9) si evince che la RAM è stabile, risulta poco occupata, raggiungendo il massimo nel caso di 100 Tx a blocco, ovvero il 25,5%, in Tab. 2 vengono mostrati i valori in percentuale dell'occupazione di RAM che, come si può vedere, non risulta essere impegnativa per il sistema.

Tabella 2: Dati uso RAM % - TS = 5s

	1 Tx	10 Tx	100 Tx
Max	22,55%	23,49%	25,48%
Min	21,62%	22,76%	23,30%
Media	22,08%	23,25%	24,51%

Parlando di memoria, si è andati quindi ad analizzare la memoria virtuale gestita dal Runtime di Besu. (Fig. 10)

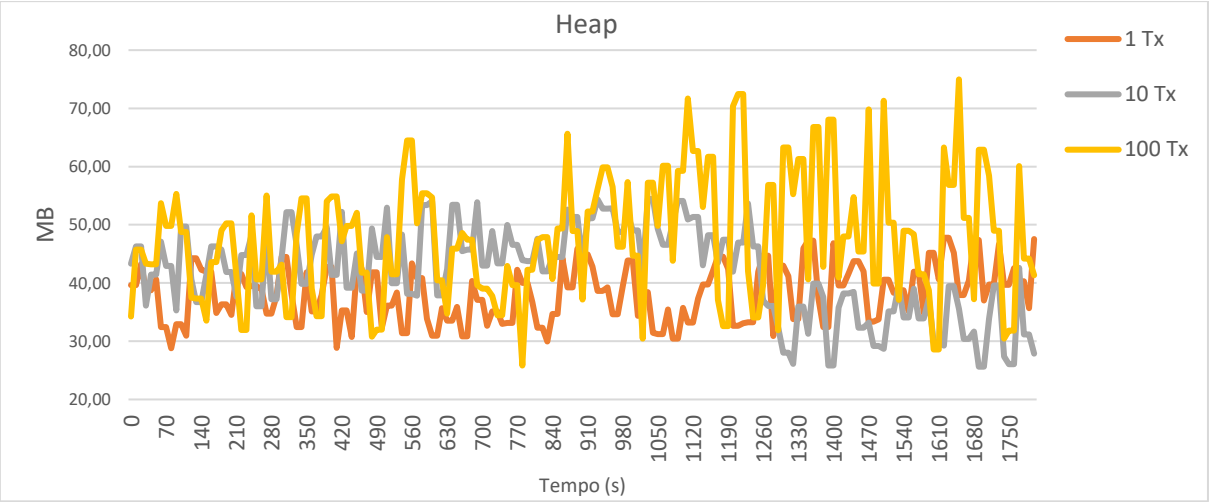

Figura 10: Memoria Heap JVM - Campionamento cicli - TS = 5s

La memoria Heap è molto volatile, e quindi si è andati a studiare il range di volatilità.

Sia dalla Tab. 3 che dal grafico in Fig.10, si nota come tra i tre casi, la differenza è minima, anche per il fatto che nella media la memoria si comporta similmente. Infatti in media tra 1 Tx e 100 Tx c'è una differenza di 11 MB. Il range di volatilità più importante lo troviamo nel ciclo di 100 Tx, pari a 40MB, dove passiamo da un minimo di 23 MB ad un massimo di 63 MB. Queste differenze, non sono impattanti ai fini dell'efficienza.

Tabella 3: Memoria Heap - MB usati durante i cicli di Tx - TS = 5s

	1Tx	10 Tx	100 Tx
Max	41,69	47,03	63,24
Min	23,60	27,28	23,24
Media	32,78	37,54	43,75

La situazione del non-Heap, non risulta particolare, l'andamento si può notare nel grafico in Fig.11.

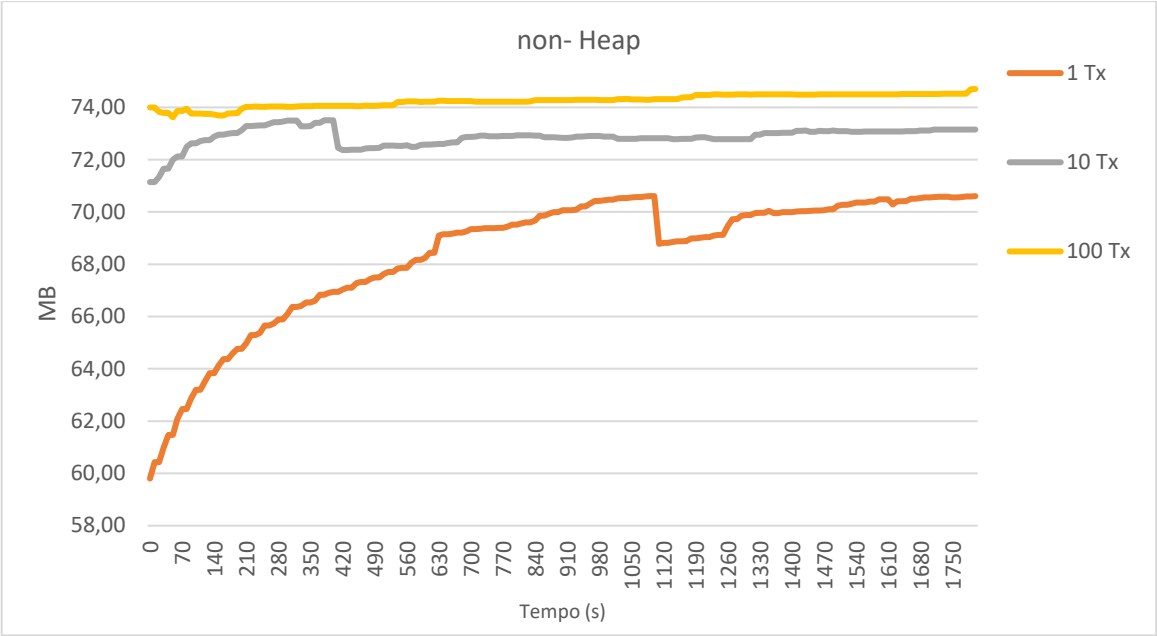

Figura 11: Memoria non-Heap JVM - Campionamento dei cicli - TS = 5s

La memoria non-heap, di per sé è stabile e quando raggiunge il regime di funzionamento non c'è molta differenza tra i 3 cicli effettuati. In Tab. 4 sono mostrati i valori della memoria non-Heap

Tabella 4: Memoria non-Heap - MB usati durante i cicli di Tx - TS = 5s

	1Tx	10 Tx	100 Tx
Max	70,61	73,51	74,70
Min	59,80	71,14	73,62
Media	68,41	72,87	74,25

Il fatto più interessante è che la memoria non-Heap, avendo un massimo di default a 64MB, al primo ciclo di transazioni (1 Tx) passa da un minimo di 60 MB ad un massimo di 70MB, per poi rimanere stabile tra i 70 e i 75MB, quindi a richieste maggiori, la memoria JVM tende ad adattarsi.

La situazione più impattante, nel sistema in esame, è lo studio del file system (Fig.12), poiché, studiando l'andamento del file system, si può capire quanti dati effettivamente vengono immagazzinati, nell'Hard Disk e quanto spazio viene occupato.

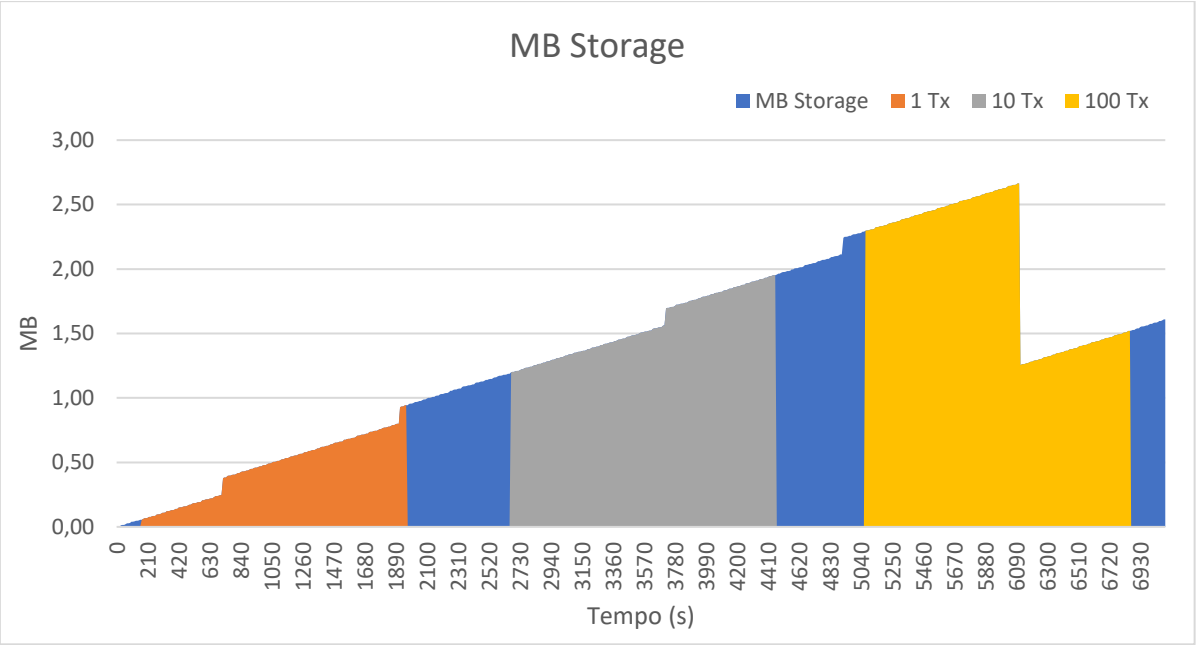

Figura 12: MB immagazzinati dal file system - TS = 5s

Dopo 3 cicli vengono immagazzinati nel totale circa 1,5 MB; nonostante durante il ciclo da 100 Tx si è arrivati anche a superare i 2,5 MB, verso la metà del ciclo si sono recuperati quasi 1,2 MB.

Nel dettaglio (Fig. 13) vediamo che l'andamento di tutti i cicli è stabile ed è molto simile, nel ciclo da 100 Tx c'è un piccolo recupero di MB, poi continua a crescere similmente alle altre due curve. Con rapido calcolo, si viaggia a circa 419 B/s scritti sull'HardDisk.

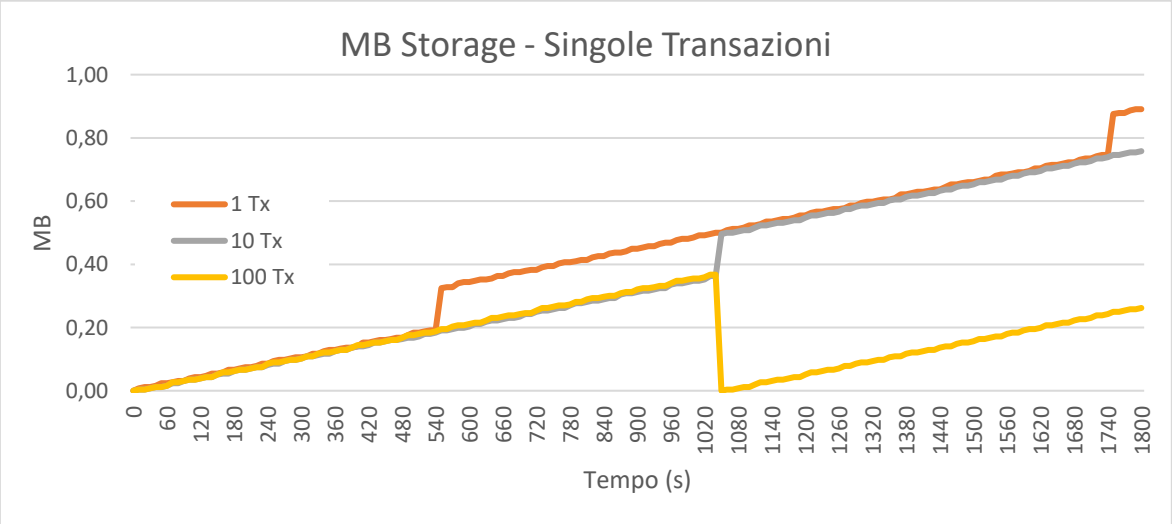

Figura 13: MB immagazzinati – Campionamento dei cicli - TS = 5s

4.2 SECONDO TEST: TS A 1 SECONDI

Nel secondo Test, si è andati a modificare solo il "TS", ovvero il "Tempo di convalida" del blocco, passando da 5 secondi a 1 secondo. La rete riceverà più dati da gestire perché diminuisce il tempo di convalida del blocco ed aumenta il numero dei blocchi da scrivere.

Con il TS a 1 secondo, la rete ha atteggiamenti diversi in base al numero di transazioni che si sta inviando. Dalla Fig. 14, si può notare che al ciclo da 1 Tx, e al ciclo da 10 Tx la rete si comporta in modo abbastanza stabile, con leggeri ritardi sull'inserimento delle transazioni nel blocco.

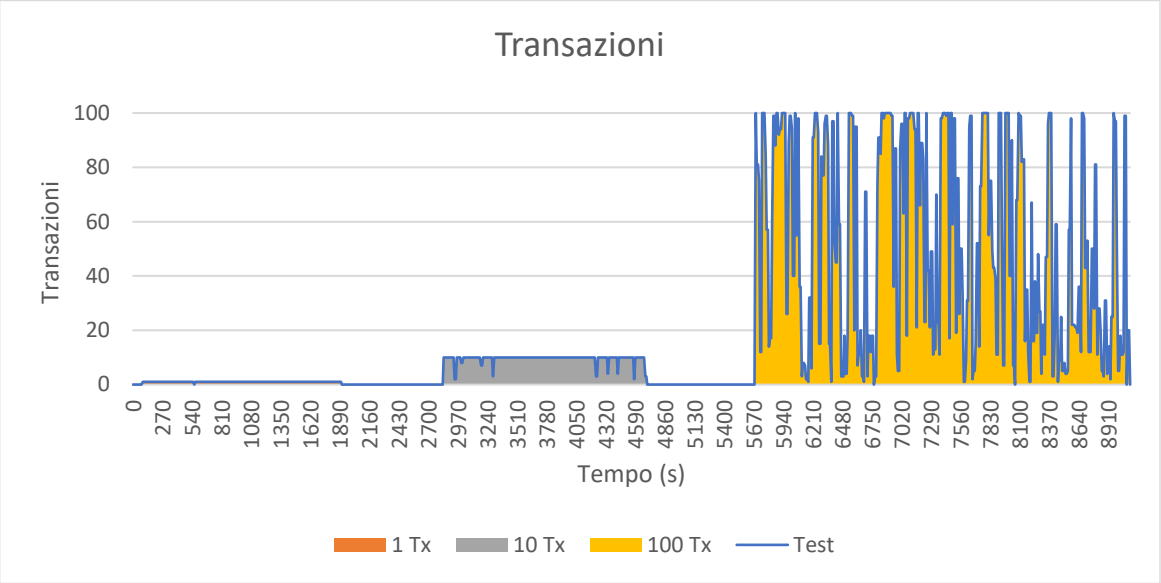

Figura 14: Transazioni per blocco – TS = 1s

Differentemente, a 100 transazioni a blocco la rete non riesce a completare il ciclo di invio delle transazioni ed è risultata instabile allungando i tempi, per cui il ciclo non si è concluso nella classica mezz'ora, bensì il tempo si allungato. Questo perché le transazioni non sono andate perse, ma vengono proposte al blocco successivo, causando così un rallentamento di Tx a blocco.

Di seguito viene studiato l'uso in percentuale della CPU durante i cicli delle transazioni.

La situazione segue l'andamento delle transazioni, dove a maggiori transazioni da gestire, maggiore è il calcolo computazionale (Fig. 15).

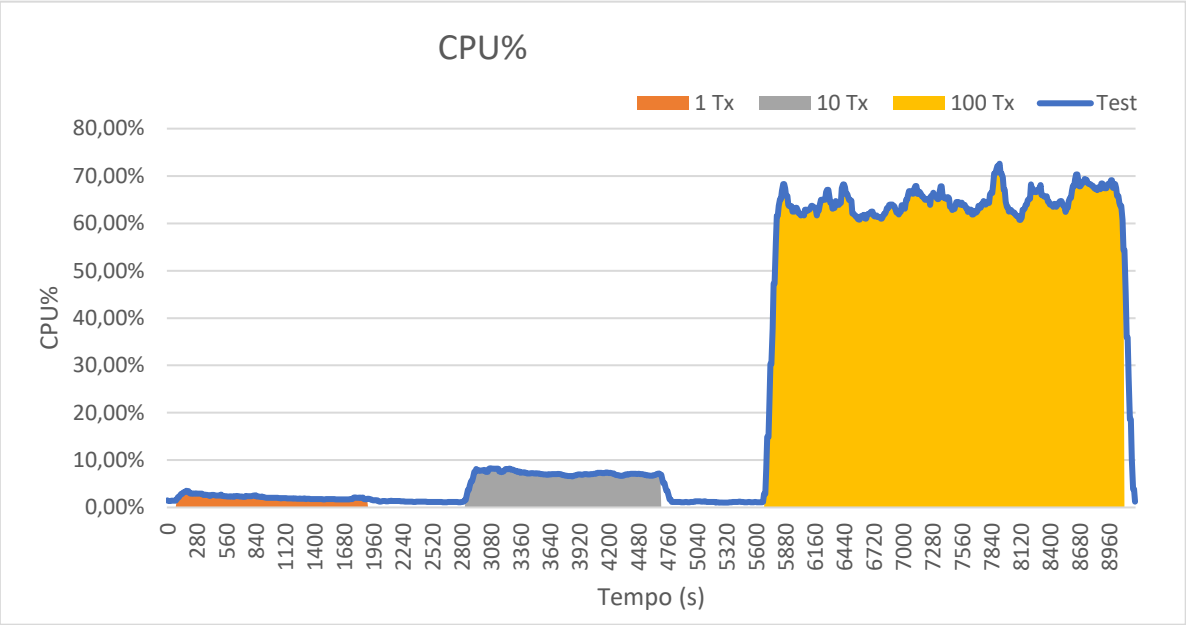

Figura 15: Uso della CPU% - TS = 1s

Campionando (Fig. 16) si è notato come effettivamente il carico sulla CPU cresca in base al numero di transazioni che si sta inviando.

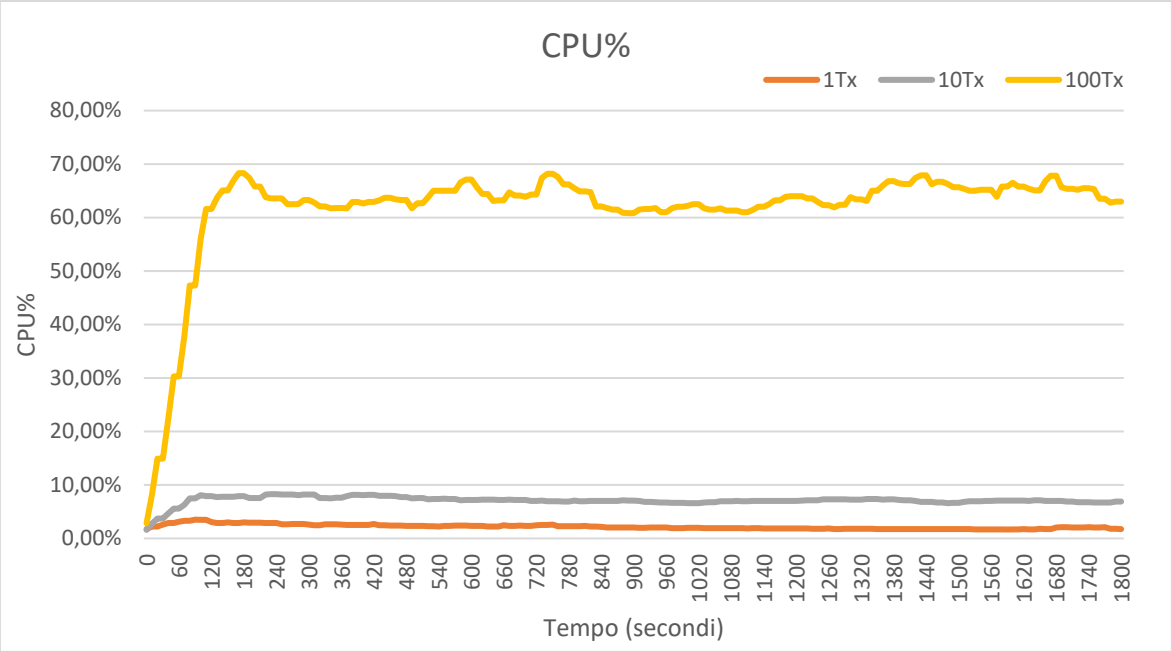

Figura 16: Uso della CPU% - Campionamento dei cicli - TS = 1s

I valori riportati nelle Tab. 5 fanno capire come viene effettivamente usata la CPU in base ai cicli precedentemente definiti.

Tabella 5: Dati uso CPU% - TS = 1s

	1 Tx	10 Tx	100 Tx
Max	3,51%	8,28%	68,30%
Min	1,67%	1,60%	2,83%
Media	2,19%	7,11%	61,85%

La situazione peggiore si conferma il ciclo di 100 Tx dove in media ci si assesta al 60% con andatura molto instabile in confronto alle precedenti sessioni.

Dopo l'analisi della CPU è stata analizzata la memoria RAM.

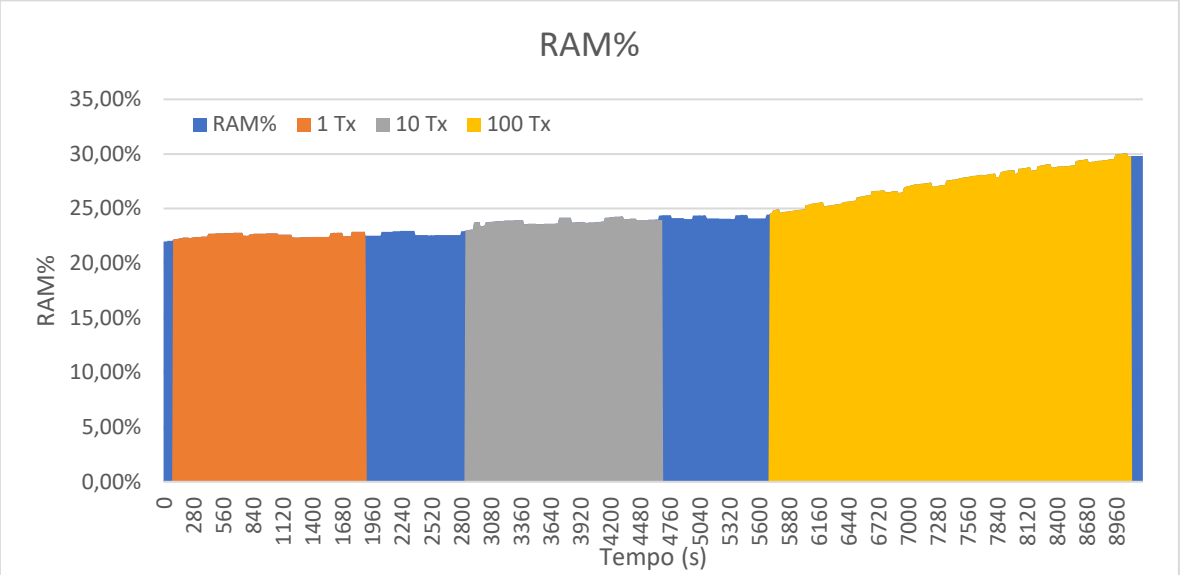

Figura 17: Uso RAM% - TS = 1s

Dall'inizio alla fine dei cicli, come mostrato in Fig. 17, la memoria RAM non risulta satura.

Analizzando nel dettaglio (Fig. 18) si nota che, in base al numero di transazioni contemporanee non c'è molta differenza.

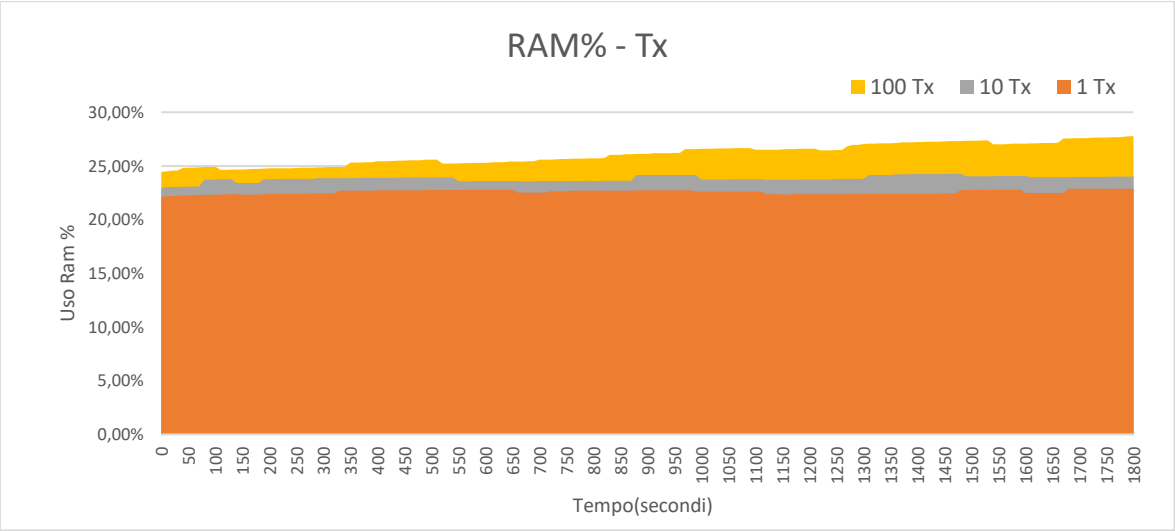

Figura 18: Uso RAM % - Campionamento dei cicli - TS = 1s

La differenza è più evidente nel momento in cui vengono inviate 100 Tx, infatti nel periodo di campionamento la memoria raggiunge un massimo di occupazione del 30%. In media, in tutti e tre i casi ci si assesta tra il 20% e il 28%. La Tab. 6 mostra i valori di occupazione della RAM.

Tabella 6: Dati uso RAM% - TS = 1s

	1 Tx	10 Tx	100 Tx
Max	22,89%	24,27%	27,77%
Min	22,14%	22,95%	24,46%
Media	22,57%	23,82%	26,11%

Passando ad analizzare la memoria usata dalla macchina Virtuale.

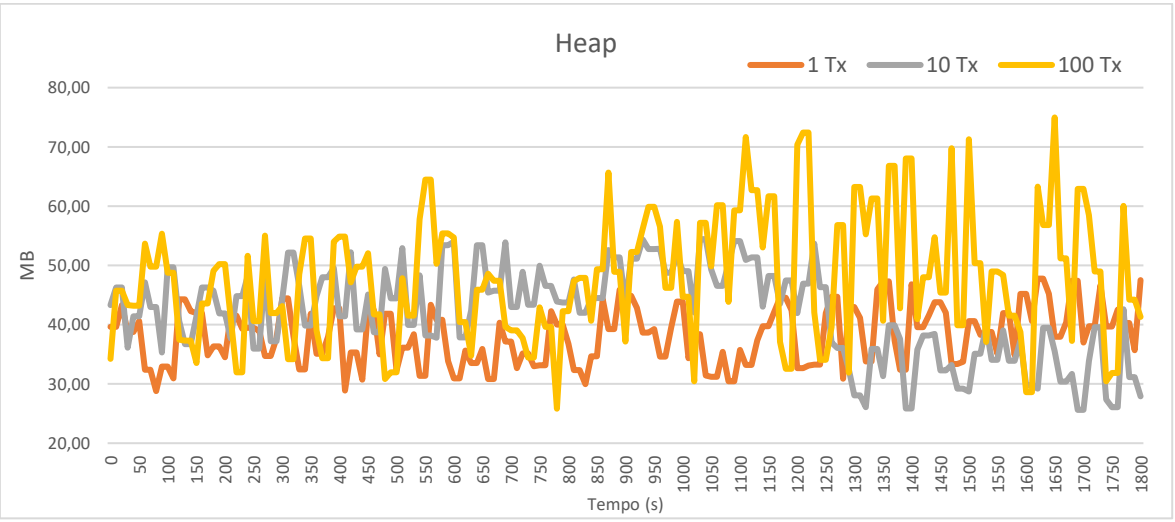

Figura 19: Memoria Heap JVM – Campionamento dei cicli - TS = 1s

Tabella 7: Memoria Heap - MB usati durante i cicli di Tx - TS = 1s

	1Tx	10 Tx	100 Tx
Max	47,78	54,44	74,97
Min	28,79	25,63	25,82
Media	38,21	41,97	47,98

Sia dalla Fig. 19 che dalla Tab. 7, la differenza notevole nel la si incontra sempre a 100 Tx, dove da un minimo di 25 MB si è passati ad avere un massimo di 74 MB, quindi un distacco di circa 50MB.

Per quanto riguarda la memoria non-Heap (Fig. 20), la situazione, in tutti e tre i casi, è stabile.

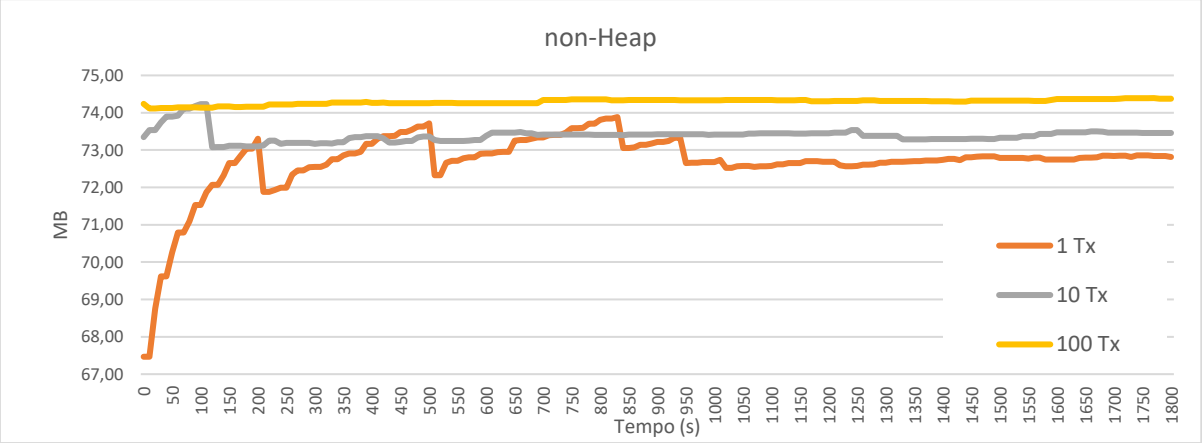

Figura 20: Memoria non-Heap JVM – Campionamento dei cicli - TS = 1s

In Fig. 21 viene mostrato come agisce il file system e quanti MB vengono effettivamente scritti.

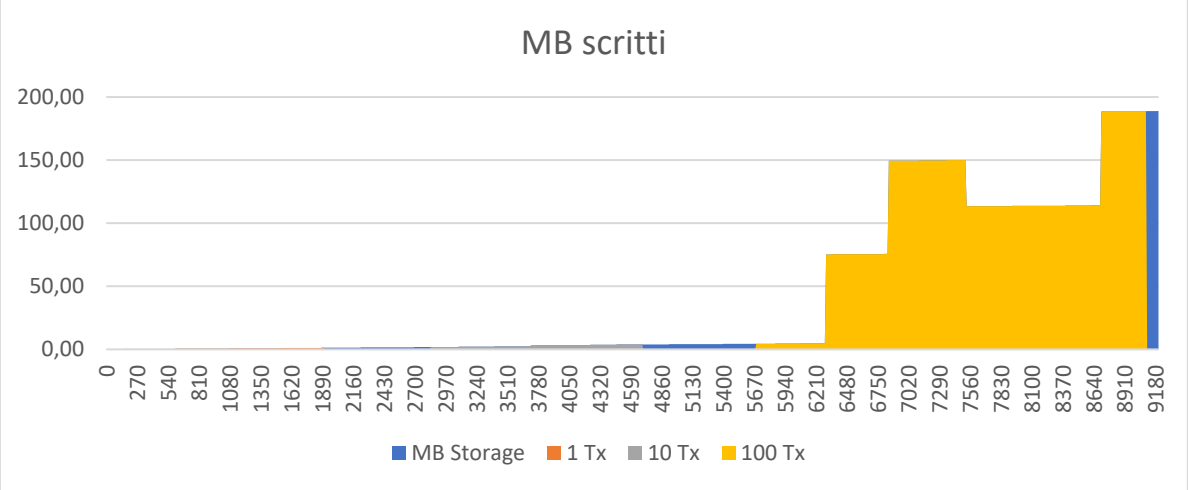

Figura 21: MB immagazzinati dal file system - TS = 1s

Si nota come l'andamento sia discontinuo solo nel caso delle 100 Tx a blocco, dove si ha anche una riduzione di memoria nello storage del file system. In questo caso durante il ciclo vengono scritti quasi 190 M.

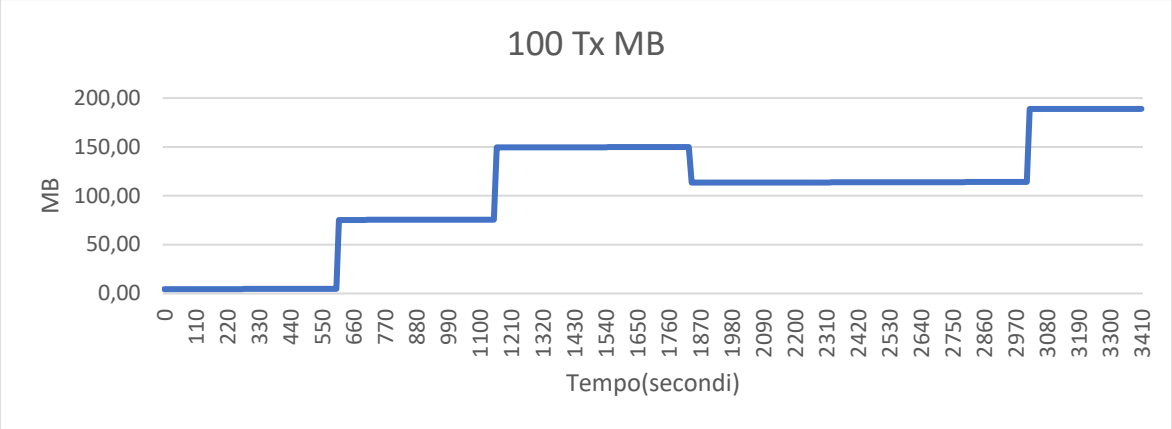

Figura 22: MB immagazzinati – Campionamento ciclo 100 Tx - TS = 1s

4.3 CONFRONTO TRA TEST IN VIRTUAL MACHINE

In entrambi i test la situazione più difficile da gestire riguarda soprattutto il ciclo a 100 Tx, infatti in tutte e due le prove, a maggior numero di Tx corrispondeva maggior uso di risorse, aggiungendo che più si abbassava il TS, maggiore era l'instabilità della rete, infatti analizzando le Transazioni per blocco si può notare, che a TS a 5 secondi la rete completava il ciclo nel tempo prestabilito, invece con il TS a 1 secondo i tempi si sono doppiamente allungati (Fig. 23).

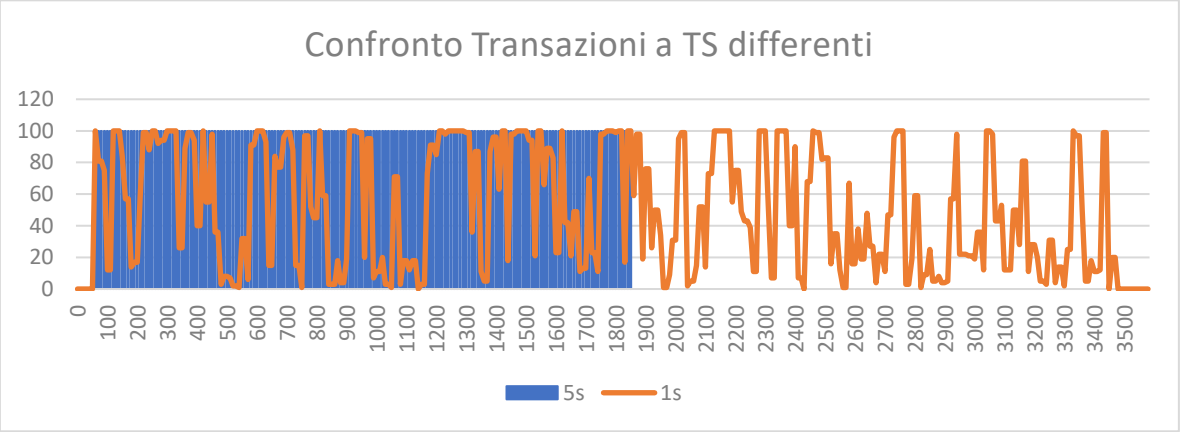

Figura 23: Confronto di Transazioni a blocco tra TS analizzati

Passando alla CPU, confrontando le medie di utilizzo, si può notare come tra il TS a 1 secondo e il TS a 5 secondi ci siano differenze importanti, analizzando solo il caso peggiore (100 Tx) l'uso del processore raggiunge in media il 60% a TS a 1 secondo, mentre raggiunge circa il 40% nel caso TS a 5 secondi. Questo dimostra che la CPU all'aumentare delle transazioni e al diminuire del TS tende ad esser più richiesta. La Tab. 8 mostra il confrontro tra stessa macchina ma TS differenti.

Tabella 8: Confronto uso della CPU% tra TS analizzati

	TS = 5s	TS = 1s
Max	51,10%	68,30%
Min	0,80%	2,83%
Media	12,40%	61,85%

La RAM non risente delle transazioni, non arriva ad una saturazione tale da essere sovraccaricata. C'è però una piccola differenza per quanto riguarda le due modalità di test, si ha un uso della memoria del 30% nella situazione TS a 1 secondo, differentemente all'altra situazione dove l'occupazione della RAM arriva a non poco oltre il 25%.

Interessante è la differenza della memoria Heap tra massimo e minimo nei test a differente TS analizzati. In Tab. 9 vengono comparati i dati dei cicli a differenti TS.

Tabella 9: Confronto Memoria Heap - MB usati nei cicli di Tx tra TS analizzati

	1 Tx		10 Tx		100 Tx	
	TS = 5s	TS = 1s	TS = 5s	TS = 1s	TS = 5s	TS = 1s
Max	41,69	47,78	47,03	54,44	63,24	74,97
Min	23,60	28,79	27,28	25,63	23,24	25,82
Media	32,78	38,21	37,54	41,97	43,75	47,98

Questo dimostra che, più il tempo di convalidazione si abbassa e maggiore è il numero di transazioni, più per la memoria Heap aumenta il range di differenza tra massimo e minimo, quindi la memoria Heap è una risorsa molto usata.

Il caso invece della memoria non-Heap non è molto differente nei due casi. Di default, una JVM, viene impostata a 64 MB. In entrambi i casi, anche nel ciclo ad una transazione, la richiesta è maggiore, e quindi la JVM aumenta la sua memoria non-Heap di pochi MB raggiungendo i 75MB. Probabilmente, a maggior numero di transazioni, ci sarà maggiore richiesta.

Per quanto riguarda l'Hard Disk, il test era stato progettato per circa 1 ora e 40 minuti. Nella situazione TS a 1 s vista, l'instabilità del sistema a 100 Tx, la durata è andata ben oltre, raggiungendo quasi le 2 ore e 25 minuti. Al contrario, nella situazione a 5 secondi, il test ha rispettato il range temporale.

Nella situazione a 5 secondi, sono stati elaborati circa 88 MB nel giro di 1 ora e 40 minuti; al contrario, nella situazione ad 1 secondo, sono stati elaborati 500 MB nel giro di 2 ore e 25 minuti.

Per quanto riguarda però gli effettivi dati memorizzati, il file system registra un totale di 180 MB per il caso a TS 1, contro i 2,50 MB nel caso a TS 5. Nel caso ad 1 secondo, i dati scritti nel file system riguardano principalmente il ciclo a 100 transazioni. Ciò non desta stranezze, i dati memorizzati nei blocchi aumentano e nel breve periodo di convalida vengono validati 1800 blocchi a differenza del caso a 5 secondi dove vengono convalidati solo 360 blocchi.

4.4 TEST SU SERVER: TS A 1 SECONDO

Per le ultime fasi di test è stato predisposto un Server, per studiare una situazione di pratico interesse. Si è predisposto solo il test con TS a 1 secondo

Il protocollo di consenso usato è clique.

Nella Fig. 24 vediamo i cicli di transazione:

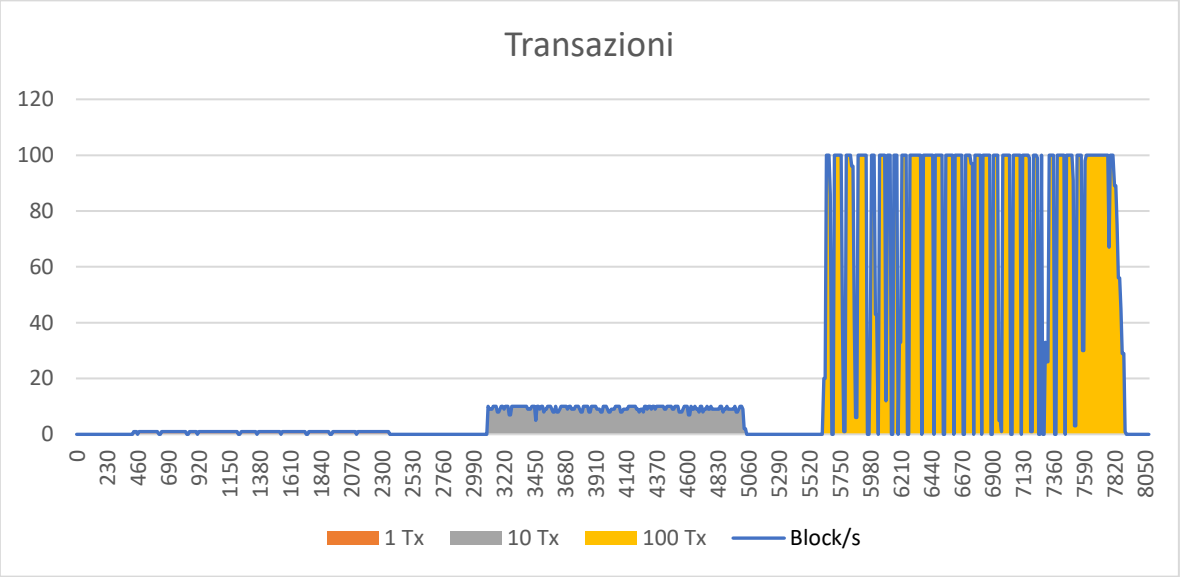

Figura 24: Server: Transazioni (TS = 1s)

Come era prevedibile il ciclo da 100 Tx risulta più instabile.

Di seguito (Fig.25) l'uso in percentuale della CPU durante i cicli delle transazioni.

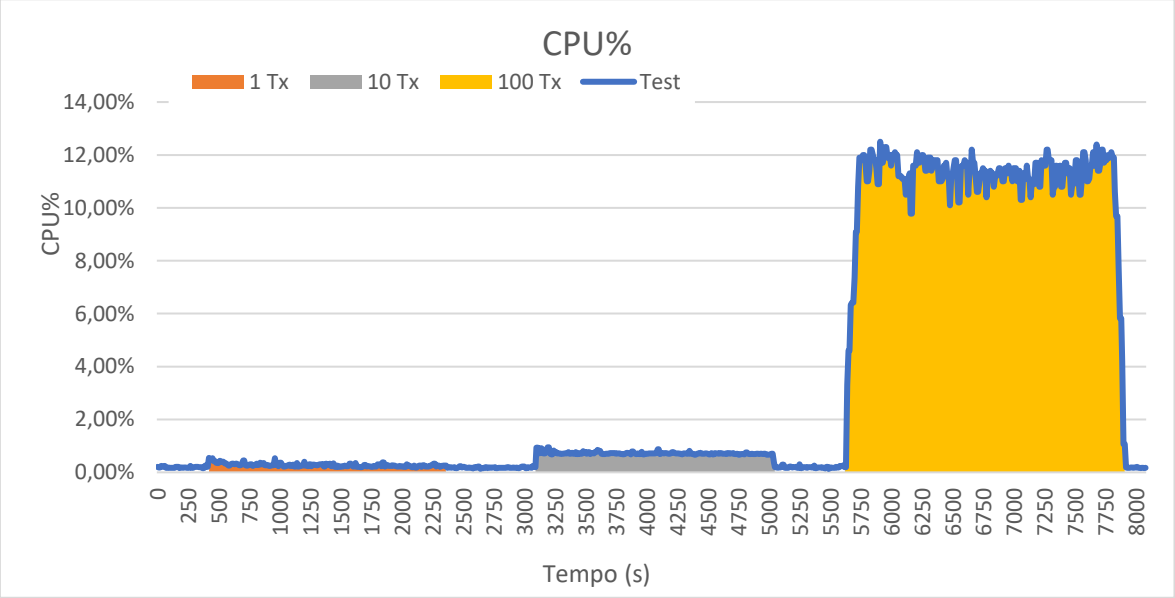

Figura 25: Server - Uso CPU% - TS = 1s

Campionando (Fig. 26) si è andati a notare come effettivamente il carico sulla CPU cresca in base al numero di Transazioni che si stanno inviando. Come comunque era prevedibile

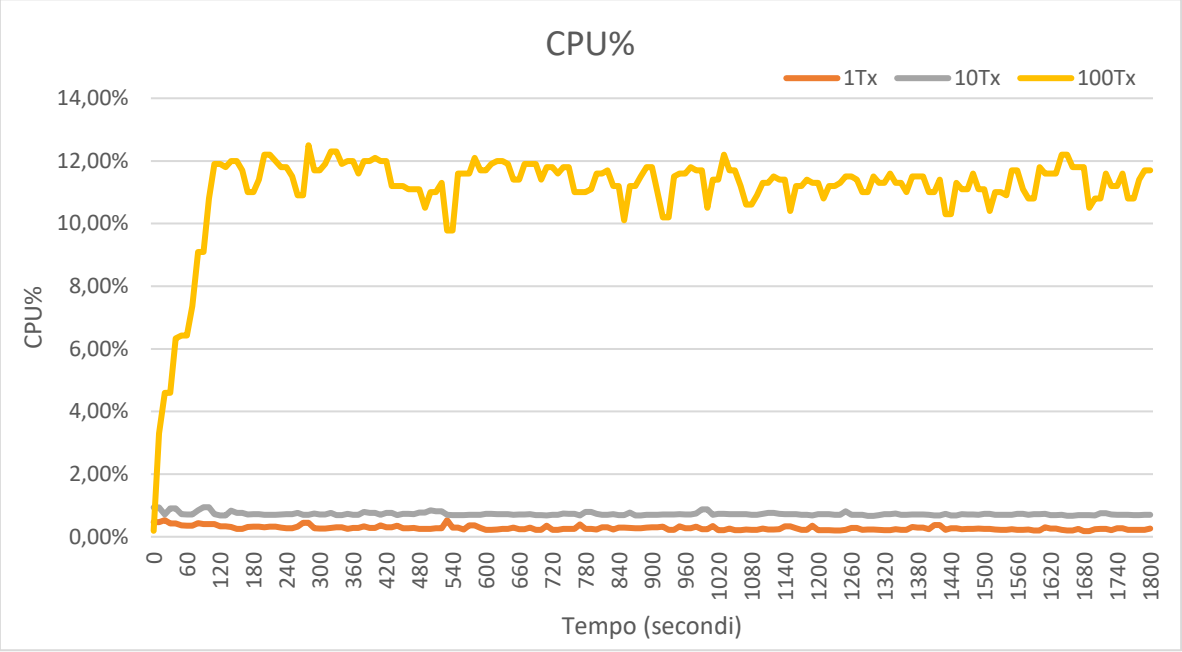

Figura 26: Server - Uso CPU% - Campionamento delle transazioni - TS = 1s

I valori, riportati in Tab. 10 fanno capire come venga effettivamente usata la CPU in base ai cicli precedentemente definiti.

Tabella 10: Server - Dati Uso CPU% - TS = 1s

	1 Tx	10 Tx	100 Tx
Max	0,53%	0,94%	12,50%
Min	0,18%	0,67%	0,19%
Media	0,28%	0,73%	11,08%

Il server usa mostra un calcolo computazionale basso, il ciclo da 100 Tx supera il 12%.

Dopo l'analisi della CPU si passa alla memoria RAM.

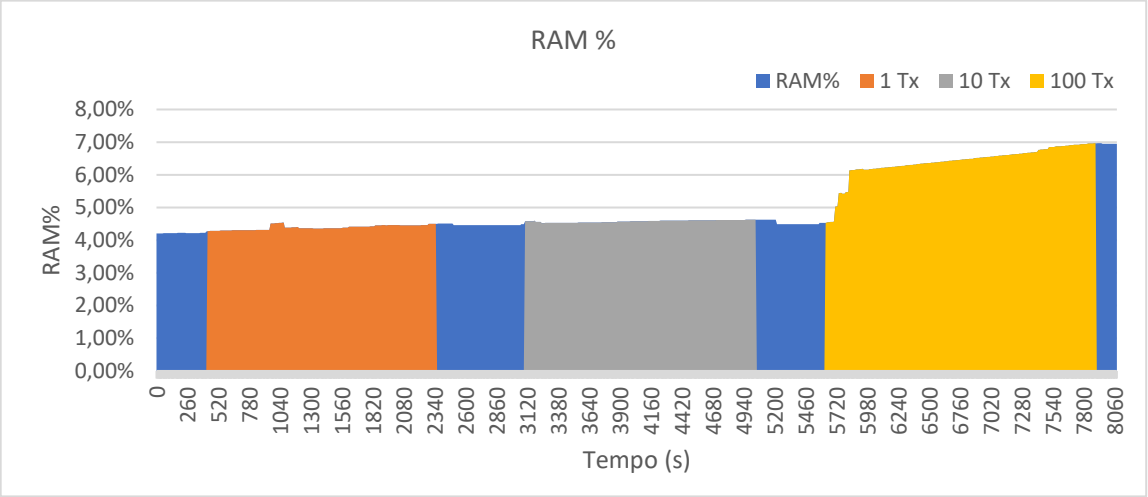

Figura 27: Server – Uso RAM% - Ts = 1s

Dall'inizio alla fine dei cicli, come mostrato in Fig. 27, la memoria RAM, rimane ben contenuta. La seguente tabella mostra che, durante il processo di gestione delle Transazioni, la RAM che viene occupata è, al massimo, circa il 7%.

Analizzando nel dettaglio (Fig. 28), si può vedere che, anche se di poco, durante le transazioni, la memoria tende ad essere impegnata, anche se in questo caso molto poco.

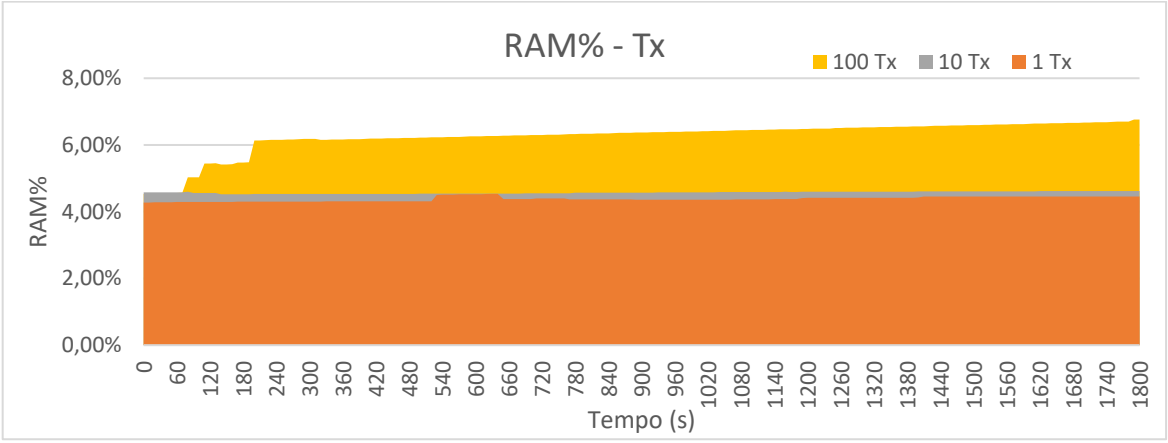

Figura 28: Server – Uso RAM% - Campionamento dei cicli - TS = 1s

Si nota che, in base al numero di Transazioni non c'è molta differenza.

Il ciclo da 100 Tx, tende ad essere quello che richiede più RAM, ma in media in tutti è tre I casi ci si assestano tra il 4% e il 7%.

Tabella 11: Server - Dati occupazione RAM% - TS = 1s

	1 Tx	10 Tx	100 Tx
Max	4,54%	4,62%	6,76%
Min	4,27%	4,52%	4,53%
Media	4,38%	4,57%	6,26%

Altro dato interessante è la memoria usata dalla macchina Virtuale. In Fig. 29 sono stati raccolti i dati riguardanti la Memoria Heap dei 3 cicli.

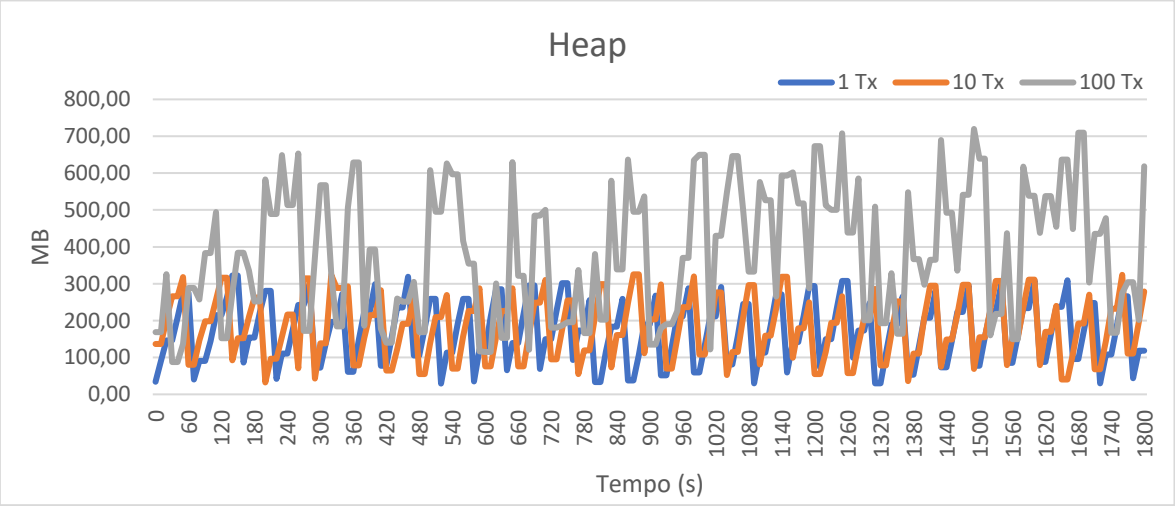

Figura 29: Server - Memoria JVM - Campionamento dei cicli - TS = 1s

Sia dalla Fig. 33, che dalla Tab. 12, si nota che non ci sono grandi differenze tra il ciclo tra 1 Tx e 10 TX. La differenza notevole la incontriamo a 100 Tx, dove da un minimo di 88 MB si può arrivare ad un massimo di 720 MB, quindi un distacco di circa 610 MB.

Tabella 12: Server - Memoria Heap - MB usati durante i cicli di Tx - TS = 1s

	1Tx	10 Tx	100 Tx
Max	322,36	325,48	719,77
Min	29,27	32,28	87,67
Media	170,67	182,08	383,43

Per quanto riguarda la memoria non-Heap, la situazione, in tutti e tre i casi è una situazione stabile:

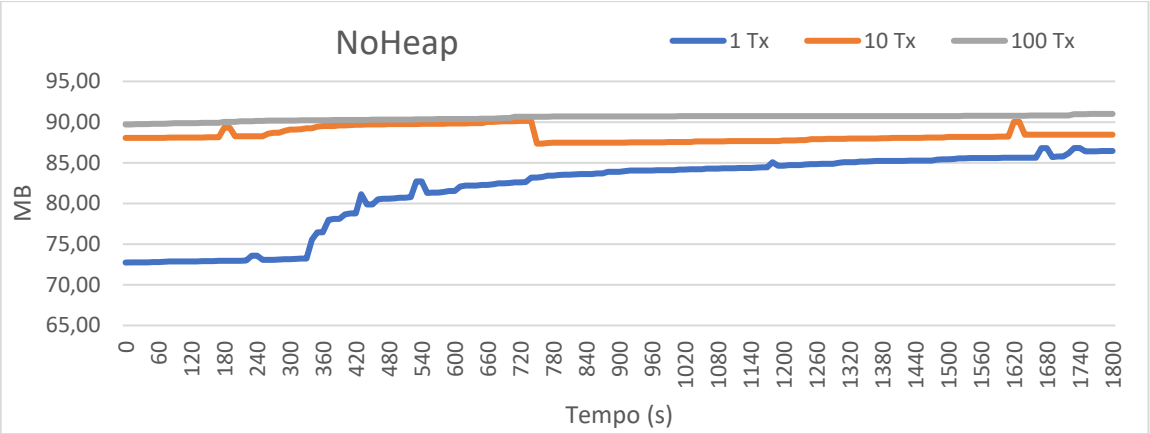

Figura 30: Server - Memoria JVM non-Heap - Campionamento dei cicli - TS = 1s

Di default, la memoria no-heap è di 64MB, in tutti e tre i casi la situazione è maggiore a 64MB, per la precisione vicino agli 88MB, con piccole differenze tra i tre casi, di circa un 3 MB.

Nel file system (Fig. 31) l'andamento è discontinuo solo nel ciclo da 100 Tx.

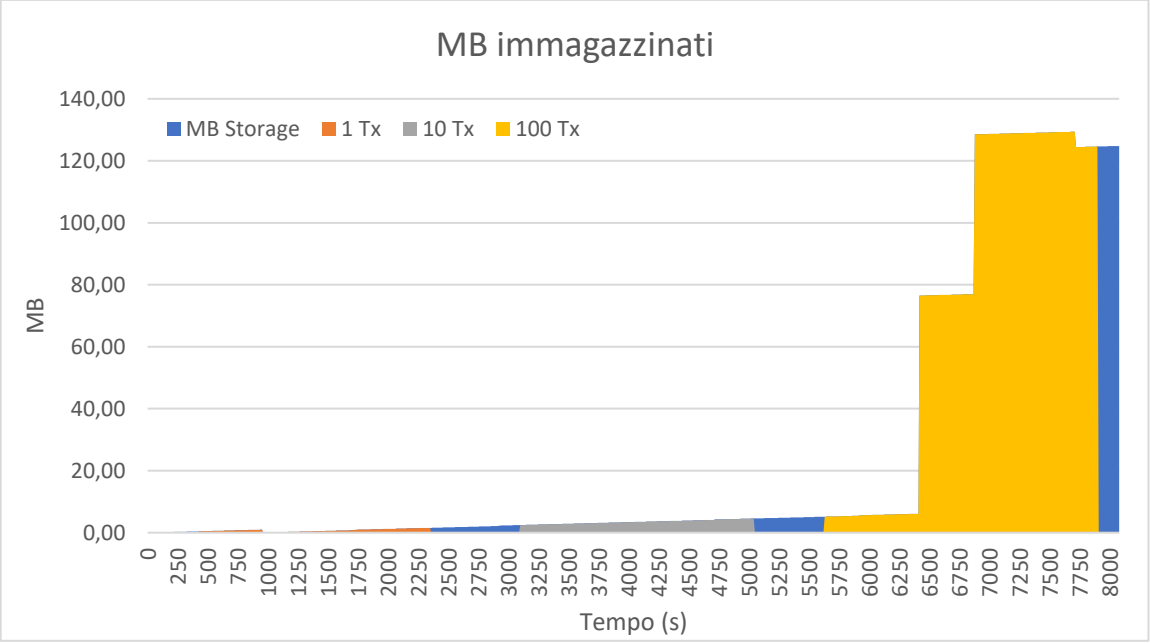

Figura 31:Server - MB immagazzinati dal file system - TS = 1s

Si nota come mentre nei primi due cicli abbiamo pochi MB scritti, analizzando, come mostrato in Fig. 32 durante il ciclo a 100 Tx, sono invece scritti 129 MB in modo molto discontinuo.

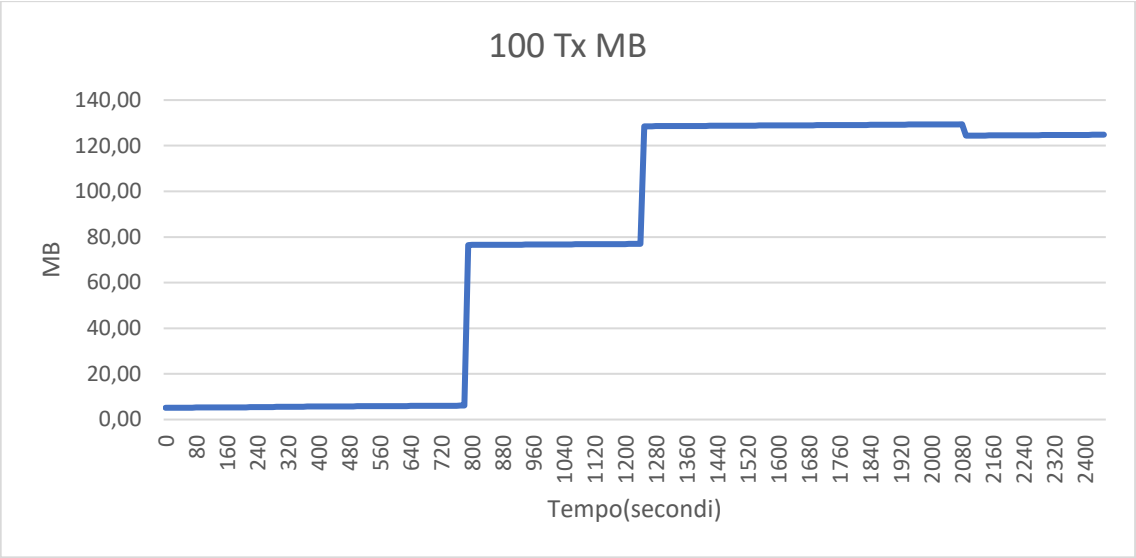

Figura 32: Server - Campionamento ciclo 100Tx - MB immagazzinati - TS = 1s

4.5 CONFRONTO TRA TEST VM E SERVER: TS 1 SECONDO

Con il tempo di validazione ad un secondo, la situazione soprattutto a 100 Tx a blocco, risulta instabile in entrambi i casi, il nodo non riesce a gestire tutte le transazioni in entrata, però si nota una certa differenza tra i due casi (Fig. 33).

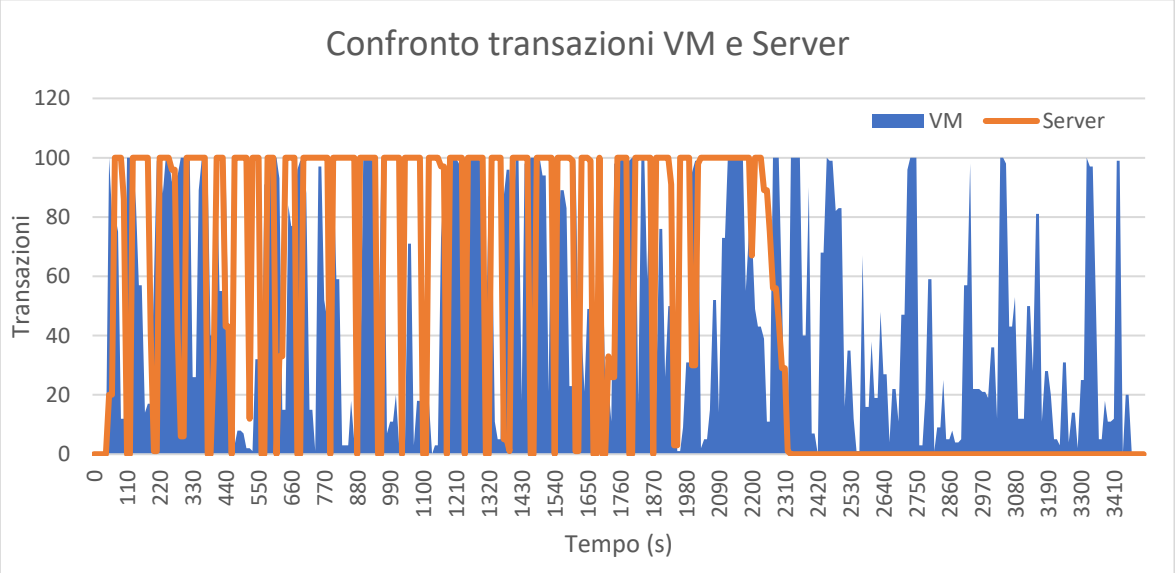

Figura 33: Confronto ciclo 100 Tx tra Server – VM

Anche calcolando la media di Transazioni a Blocco (Tab. 13) si nota come il server riesce a gestire in media 75 transazioni a blocco, mentre la macchina virtuale solo 53, quindi ben 22 transazioni in meno.

Tabella 13: Confronto media di Transazioni inviate nel ciclo 100 Tx tra Server - VM

	Media
Server	74,96
VM	53,13

L'uso del Processore nel Server vede un netto miglioramento (Tab. 14). Si passa dal 70% in macchina virtuale, ad un 12,5% di utilizzo nel Server nel caso del ciclo delle 100 Tx. Questo comportamento nell'utilizzo della risorsa era previsto che fosse migliorato poiché si passa dall'utilizzo di un solo core a 1 thread, all'utilizzo di 10 core a 20 threads

Tabella 14: Confronto uso CPU% tra Server - VM

	Server	VM
Max	12,50%	72,60%
Min	0,13%	1,04%
Media	3,39%	26,03%

In entrambi i casi, la RAM non risente delle transazioni, non arriva ad una saturazione tale da essere sovraccaricata. C'è però una piccola differenza per quanto riguarda le due modalità di test, abbiamo un uso della memoria del 30% nella situazione della macchina virtuale, differentemente all'altra situazione dove l'occupazione della RAM arriva a non poco oltre il 6,5%. Tendenzialmente la situazione in questo caso è simile alla CPU, era previsto un uso minore della risorsa, considerando anche il fatto di essere passati da 4GB a 32GB.

Il confronto in questo caso mostra un leggero differente uso della memoria non-Heap. La non-Heap ha un incremento maggiore nel Server in confronto alla Macchina Virtuale. Si passa da un incremento di circa 11 MB da default sulla Macchina Virtuale ad un incremento di quasi 27MB sul server. (75 MB Macchina Virtuale – 91 MB sul Server), l'incremento però sul Server è visibile già dalle prime transazioni, dove nel ciclo da 1 Tx, passiamo da 74MB a 86MB.

Il caso invece della memoria Heap è molto differente nei due casi (Tab.15). Il range di volatilità sul Server è ampissimo, già nelle transazioni più piccole si passa da 30 MB a circa 330MB, un range di quasi 300MB; la differenza più ampia però si vede nel ciclo da 100 Tx dove il range è ancora più marcato passando da un minimo di 90 MB ad un massimo di 720MB. Numeri ben lontani da i 75 MB della macchina virtuale.

Tabella 15: Confronto Memoria Heap - MB usati durante i cicli di Tx tra Server - VM

	1 Tx		10 Tx		100 Tx	
	VM	Server	VM	Server	VM	Server
Max	47,78	322,36	54,44	325,48	74,97	719,77
Min	28,79	29,27	25,63	32,28	25,82	87,67
Media	38,21	170,67	41,97	182,08	47,98	383,43

Questo dimostra che, data la possibilità di accesso a risorse maggiori e migliori, la JVM tenta di ottenere il massimo, e infatti il fatto che riesca a gestire più transazioni al secondo rispetto ad una macchina virtuale porta ad un incremento dell'uso della memoria virtuale da parte della JVM stessa. Lo stesso test su due macchine diverse non ha evidenziato grandi differenze sull'Hard Disk, come per la macchina virtuale, il ciclo evidentemente più difficile da gestire è stato quello da 100Tx. Nel caso della macchina virtuale 190MB saranno quelli immagazzinati nel file system, nel server invece vengono memorizzati nel 130MB circa.

La situazione è simile, ma il server riesce ad elaborare i dati in modo migliore, Besu ha il file system riesce a comprimere meglio i dati visto che sono stati scritti 60MB in meno. Il test però ha dimostrato che, nel confronto, il Server è riuscito a risolvere il ciclo in meno tempo per cui la quantità di dati scritta è minore poiché sono stati convalidati meno blocchi.

Questo significa che in base al numero di blocchi, e alle transazioni da immagazzinare i dati scritti crescono proporzionalmente.

4.6 TEST SU SERVER CON BLOCKCHAIN A PIU' NODI: 100 TX

Una volta conclusi i test preliminari si è andati a monitorare la rete in una situazione reale, aggiungendo nodi alla rete e cambiando protocollo di consenso.

Assieme all'Università di Camerino si è avviata una rete di test composta da 7 nodi, con il protocollo di consenso IBFT2.0. Il Time Signature in questo è stato impostato a 20 secondi.

Visti i risultati da 1 Tx a 10 Tx sia su Virtual Machine che su Server, si è deciso di effettuare il test da 100 Tx a blocco, poiché è stato quello che ha risultati importanti in confronto agli altri due test, che hanno mostrato, invece, risultati simili tra di loro.

Come mostrato in Fig. 34, la situazione è molto stabile, risultato prevedibile visto il TS impostato molto lungo rispetto ai test precedenti.

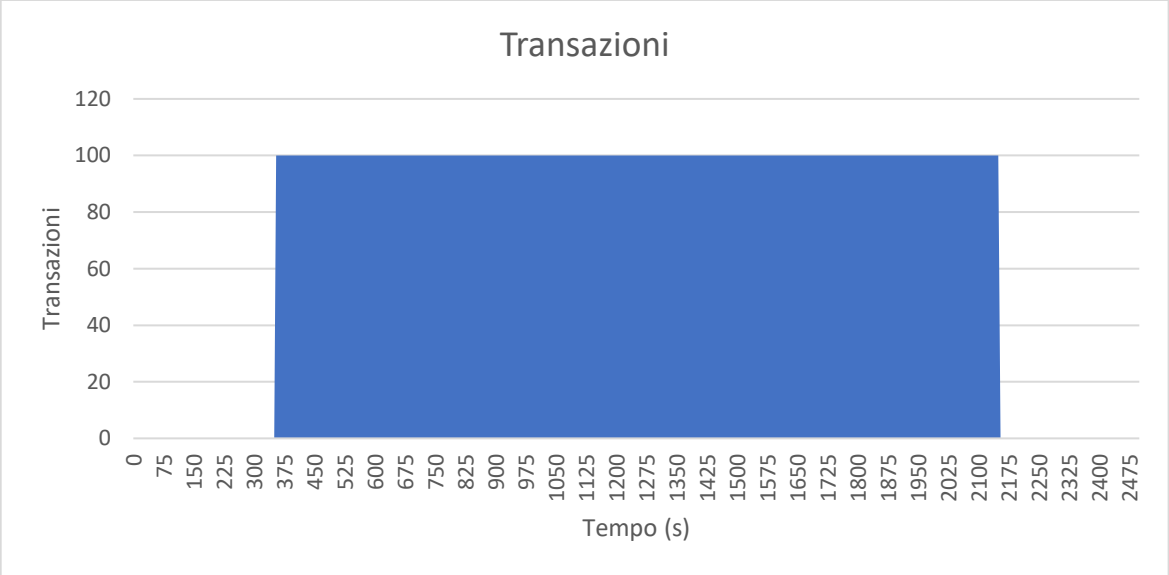

Figura 34: IBFT 2.0 - Transazioni a blocco - 100 Tx

Monitorando la CPU (Fig. 35), si nota che c'è un leggero aumento della CPU % durante le transazioni, ma nulla che renda impossibile il processo computazionale della validazione (o della ricezione) dei blocchi nel nodo.

Le percentuali che raggiunge la CPU non superano il 2%, come mostrato in Fig. 35 e Tab. 16.

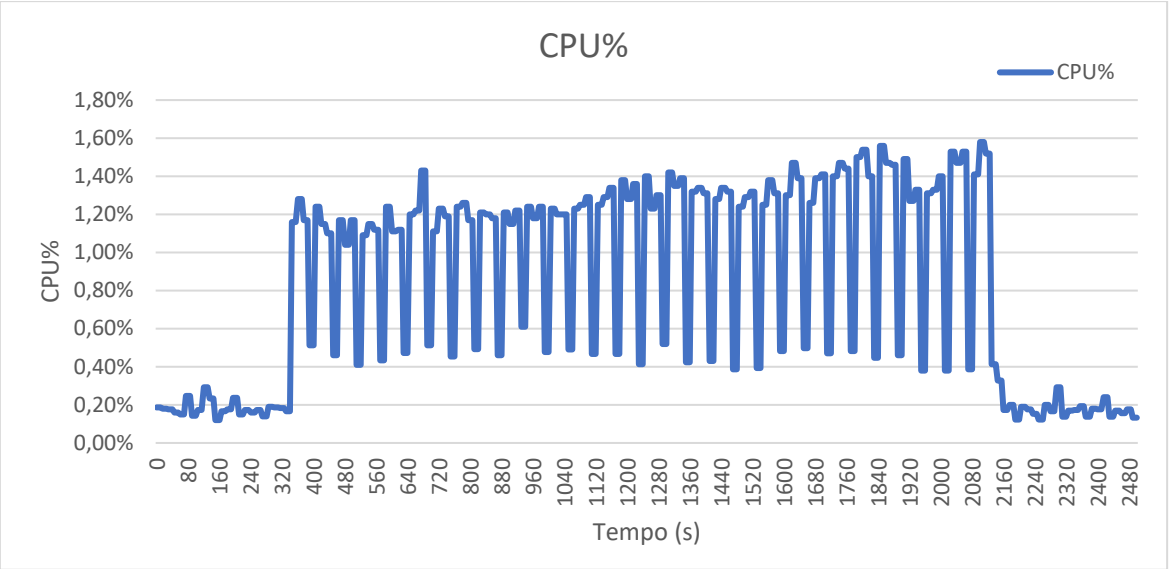

Figura 35: IBFT 2.0 - Uso CPU% - 100 Tx

Tabella 16: IBFT 2.0 - Dati Uso CPU% - 100 Tx

	CPU%
Max	1,58%
Min	0,12%
Media	0,83%

Monitorando la RAM (Fig. 36) la situazione è decisamente simile ai test precedenti.

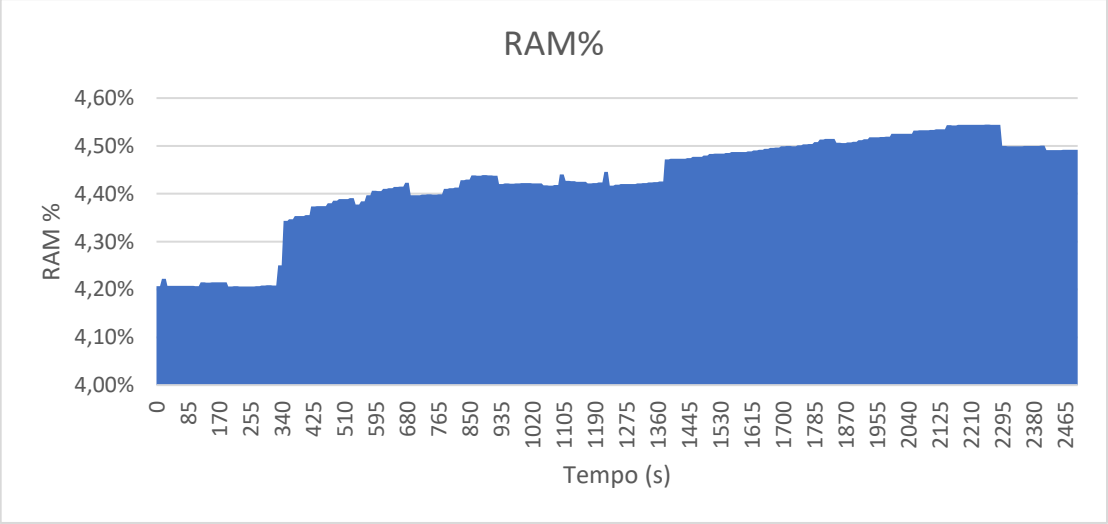

Figura 36: IBFT 2.0 - Uso RAM% - 100 Tx

Sia dalla Fig 36 che dalla Tab. 17 notiamo che l'occupazione della RAM non supera il 4,5%

Tabella 17: IBFT 2.0 - Dati uso RAM% - 100 Tx

	RAM%
Max	4,54%
Min	4,21%
Media	4,43%

Monitorando invece la Memoria della macchina virtuale, la situazione è simile al Test su Server.

Come mostrato in Fig. 37, la memoria non-Heap, tende ad essere sempre stabile e, come già visto nel caso precedente del server, è stabile a circa 96 MB.

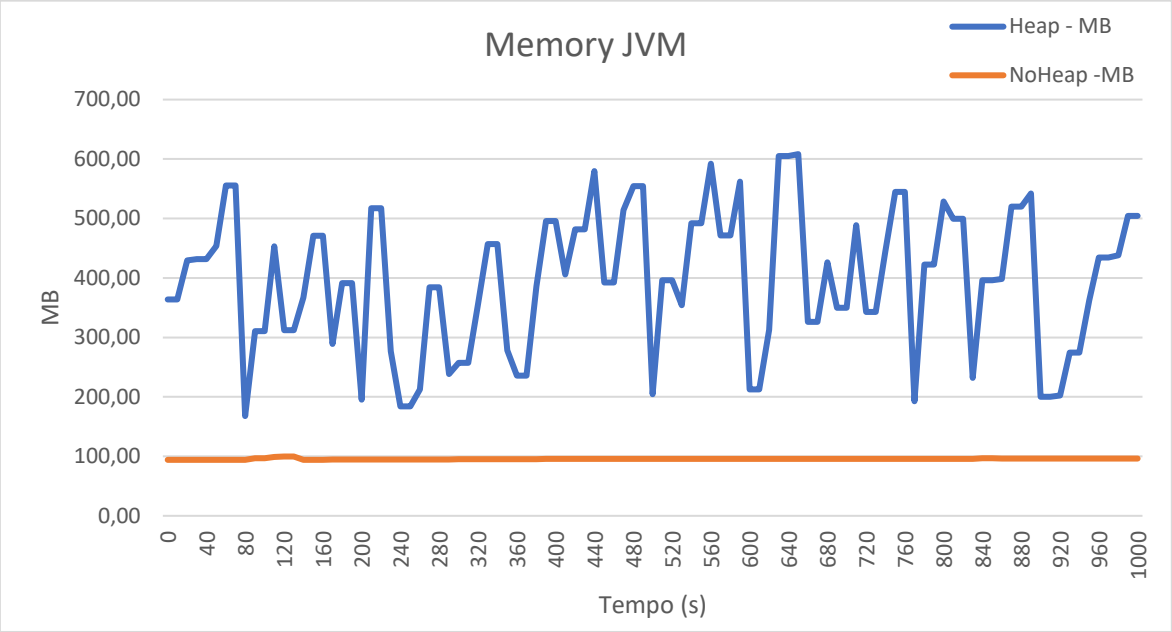

Figura 37: IBFT 2.0 - Memoria JVM - 100 Tx

La memoria Heap invece risulta molto volatile nonostante il TS molto lungo. Dalla Tab. 18si può notare che si passa da un minimo di 50 MB ad un massimo di 480 MB, un range di differenza di 430 MB.

Tabella 18: IBFT 2.0 - Memoria Heap - MB usati durante i cicli di Tx - 100 Tx

Heap	Total
Max	480,71
Min	52,66
Media	279,37

Per quanto riguarda i dati immagazzinati, si può notare (Fig. 38) come dall'inizio del ciclo fino alla fine vengono immagazzinati 2,20 MB nel filesystem.

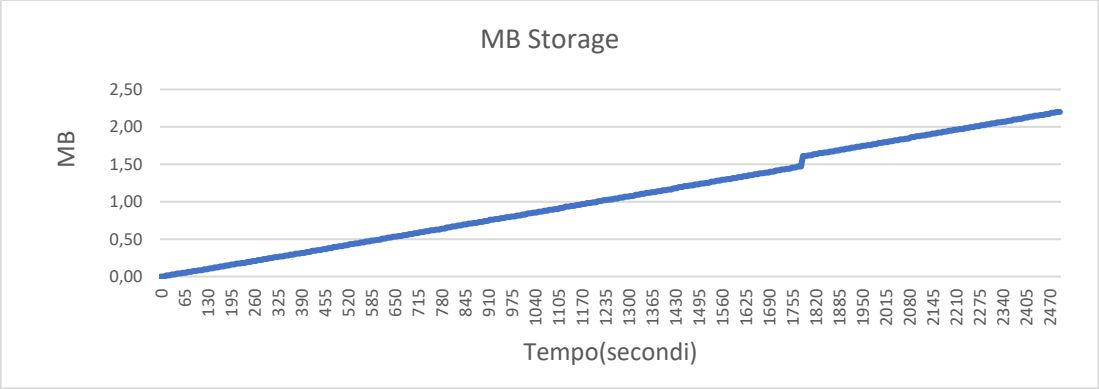

Figura 38: IBFT 2.0 - MB immagazzinati dal file system - 100 Tx

Conclusioni

Effettuato un solo ciclo, il più pesante finora effettuato, il Server e la rete IBFT 2.0 (Time Signature a 20 secondi) hanno risposto egregiamente, la CPU viene usata il meno possibile, la RAM non è satura, sia il lavoro che l'immagazzinamento dei dati risulta leggero. L'unica risorsa che lavora leggermente di più è la memoria JVM poiché, a maggiori transazioni sfrutta il più possibile le risorse che ha, nonostante il tempo di convalida lungo, il range di differenza di 430MB è minore del caso *Server, 1 TS, 100Tx*, ma molto più ampio del caso *VM, 5TS, 100Tx*, questo è dovuto dal fatto che oltre le 100Tx, si stanno elaborando anche dati in rete, poiché si sta comunicando con altri 7 nodi.

4.7 TEST SU SERVER CON BLOCKCHAIN A PIU' NODI: 500 TX

Vista l'ottima risposta, si è andati ad aumentare il numero di transazioni, passando da 100 Tx a 500 Tx.

La situazione però è risultata particolare, sul file di genesis è stato scritto un limite di gas utilizzabile, il che ha portato trascrivere solo 223 transazioni per blocco. Le transazioni in eccesso, venivano comunque accolte dalla rete, ma messe in stato di *pending*

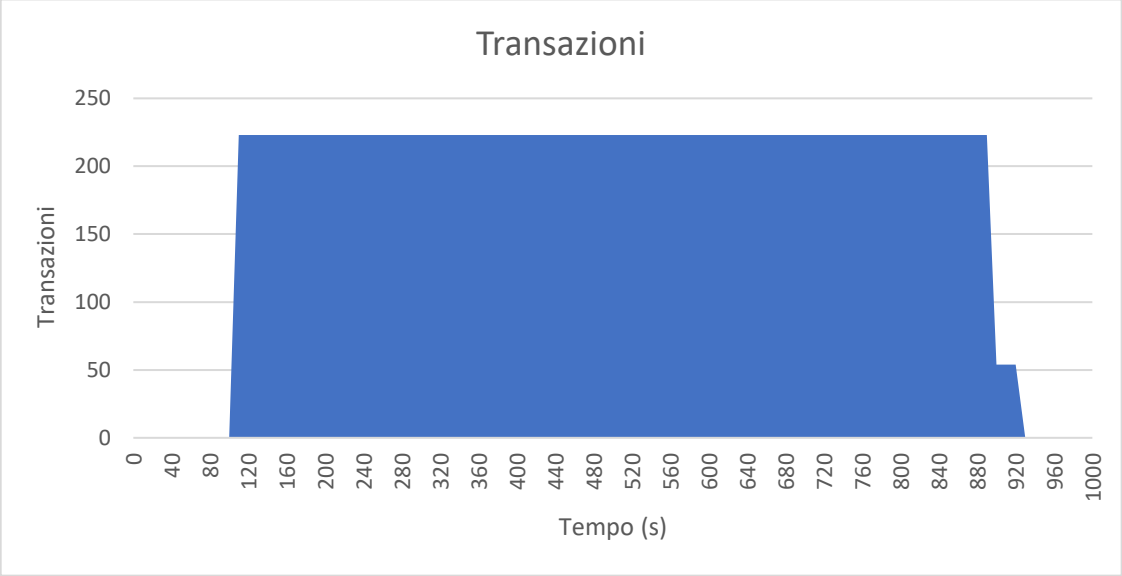

Figura 39: IBFT 2.0 – Transazioni a blocco - 500 Tx

Tuttavia la rete ha un limite temporale per le transazioni in pending, se queste entro 750 secondi non vengono effettuate vengono rigettate, ciò non ha permesso il completamento del test, poichè la rete non è riuscita a convalidare tutte le transazioni.

Per quanto riguarda le risorse, non si vedono grandi differenze.

La CPU viene usata al Massimo al 3,6%, come mostrato in Fig. 40 e Tab. 19.

Tabella 19: IBFT 2.0 - Dati Uso CPU% - 500 Tx

	CPU%
Max	3,68%
Min	0,13%
Media	1,73%

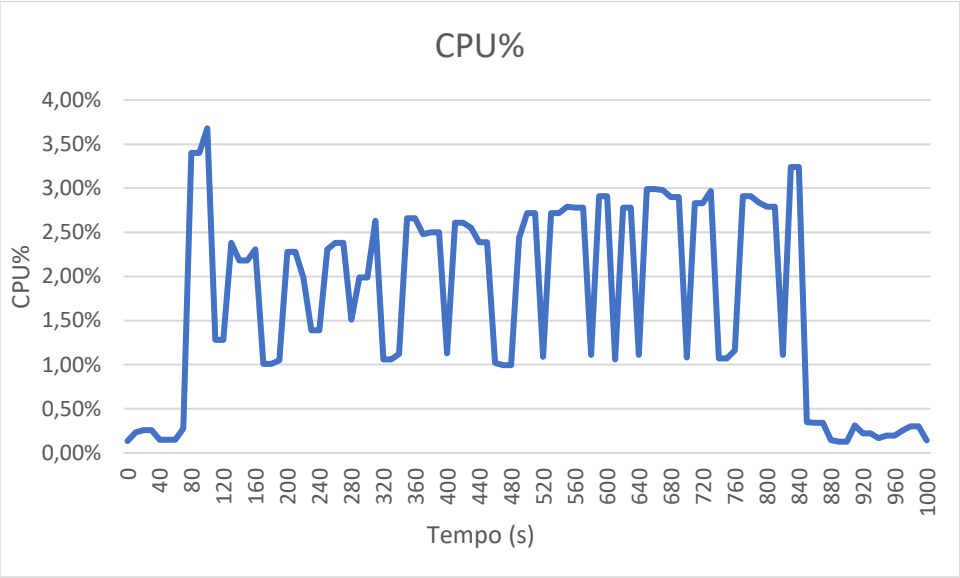

Figura 40: IBFT 2.0 - Uso CPU% - 500 Tx

Monitorando la RAM, invece, si nota un leggero incremento, ma nulla che possa saturare le prestazioni della macchina.

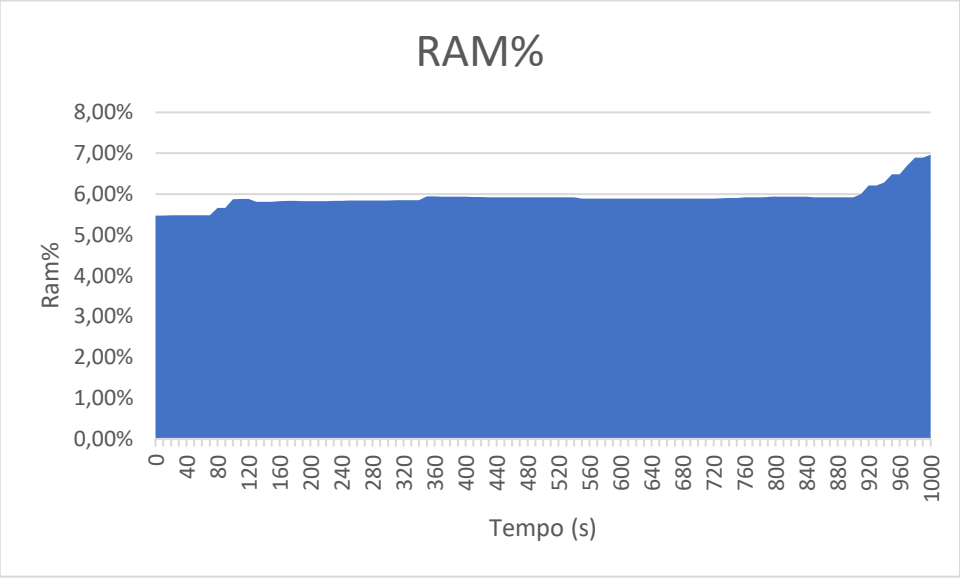

Figura 41: IBFT 2.0 - Uso RAM% - 500 Tx

Sia dal grafico in Fig. 41 che dalla Tab. 20, si nota un leggero incremento, ma di media siamo sul 6% di occupazione.

Tabella 20: IBFT 2.0 - Dati uso RAM% - 500 Tx

	RAM%
Max	7,20%
Min	5,47%
Media	6,01%

La memoria JVM (Fig. 42) si comporta similmente ai risultati precedenti, a più transazioni la memoria Heap ha un range più marcato, 440 MB di differenza come si nota dalla Tab. 21, mentre la non-Heap raggiunge un massimo di 100MB.

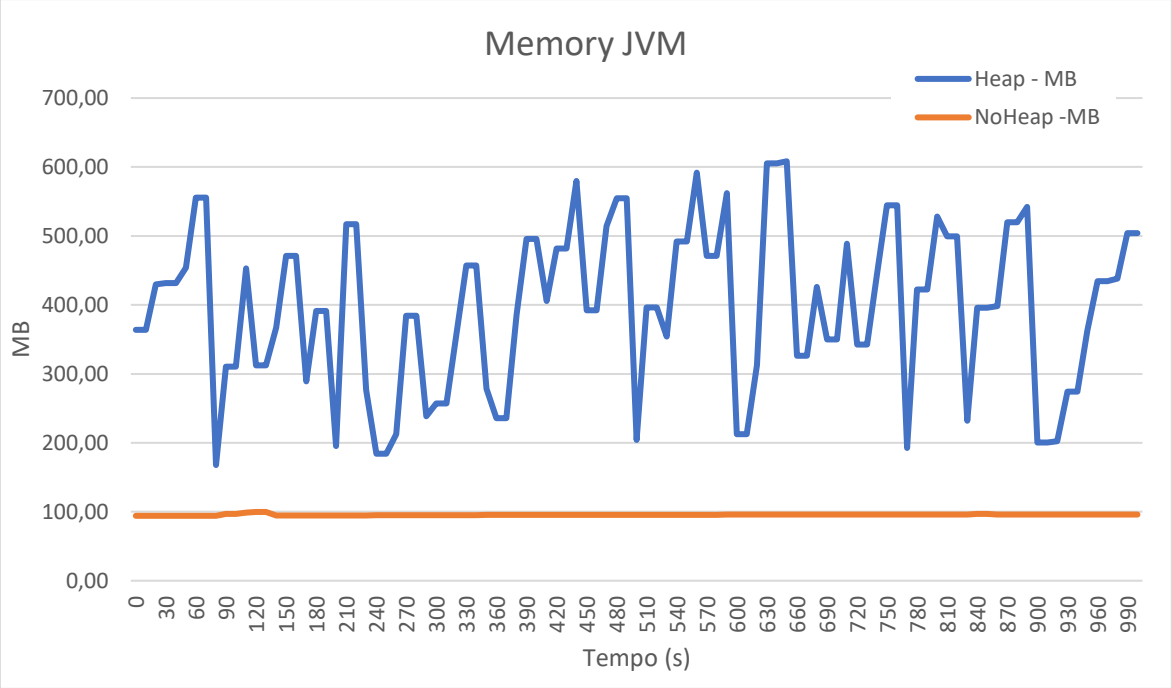

Figura 42: IBFT 2.0 - Memoria JVM - 500 Tx

Tabella 21: IBFT 2.0 - Memoria Heap - MB usati durante i cicli di Tx - 500 Tx

Heap	Total
Max	608,32
Min	167,81
Media	396,75

In questo caso l'Hard Disk, ha immagazzinato circa 800KB, come mostrato in Fig. 43. La causa è perché non ha completato il ciclo di transazioni.

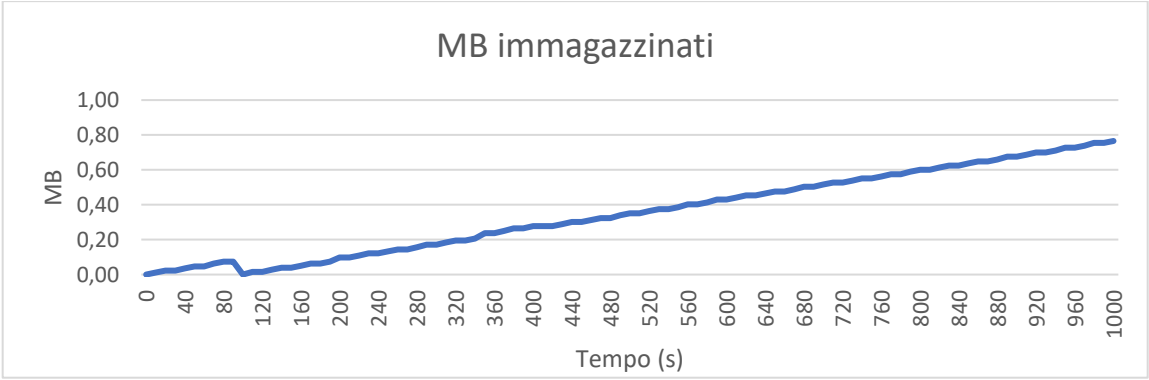

Figura 43: IBFT 2.0 - MB immagazzinati dal file system - 500 Tx

Conclusioni

La situazione a 500 Tx, comunque non ha sortito effetti diversi da quella a 100 Tx, da un lato ci sono le perfomance del Server, che hanno risposto ottimamente ai test, dall'altra c'è il limite massimo di Gas che non ha permesso un benchmark più pesante, ma comunque ha portato risultati importanti.

5. CONCLUSIONI

I risultati ottenuti hanno mostrato un'infrastruttura molto resistente e che non impatta sulle componenti hardware su cui è implementata.

Si è potuto notare, risultato prevedibile, che le risorse hardware vengono più appesantite al maggior invio di transazioni contemporanee. La situazione peggiore è risultata il ciclo da 100 Tx.

Partendo dalla CPU, già in macchina virtuale, si è potuto constatare che la risorsa è legata al numero di Transazioni e al TS; più il TS è basso e il numero di Transazioni è alto, più il processore viene sfruttato. Con l'aumento delle prestazioni il comportamento è rimasto simile.

Altro dato importante è la memoria JVM. La non-Heap non è impattante cresce con il tempo. Però, già alle prime operazioni, tende a stabilizzarsi; al contrario della Heap dove la volatilità cresce all'aumentare delle transazioni, dato interessante, è che se ha possibilità di avere maggiori prestazioni delle risorse, la Memoria JVM tende a usarne il più possibile.

Anche l'Hard Disk è stata una risorsa molto impegnata, ha lo stesso comportamento della CPU, a situazioni stabili, tende a scrivere pochi MB nel file system; all'aumentare delle Tx, l'Hard Disk ha iniziato scrivere molti più dati sul file system. C'è da sottolineare, però, che in un'ora di Transazioni, nelle situazioni peggiori, l'Hard Disk ha scritto 200MB.

La RAM invece non ha sortito effetti indesiderati, ed è sempre rimasta stabile in tutti i casi studiati, occupando poco spazio.

I casi studiati sono simulazioni di situazioni a condizione estreme, oltre il limite prestazionale di una blockchain Besu reale, situazioni selezionate per testare l'impatto sull'hardware su cui la catena è stata implementata, ma che in situazioni reali difficilmente si troverà a lavorare. Infatti, in una catena client Ethereum reale, come Besu, si ha un periodo di validazione dei blocchi molto lungo; 5 secondi ed 1 secondo sono valori usati per stressare l'hardware e valutare il comportamento della catena, il tempo medio di validazione è di circa 30 secondi in un client Ethereum reale. In più è possibile scegliere anche il limite di gas utilizzabile per blocco, così facendo si va a limitare il numero di Transazioni che il blocco può contenere.

Hyperledger Besu, è risultato comunque uno strumento, di facile configurazione e, soprattutto, interfacciamento; i risultati dimostrano che anche in una situazione molto pesante, Besu è riuscito ad ottenere buoni risultati sugli hardware implementati.

In conclusione, Hyperledger Besu è risultata un'ottima alternativa di Blockchain privata, anche in prospettiva di una sua implementazione aziendale, non a caso l'Europa, tramite il progetto European Blockchain Services Infrastructure (EBSI) [14], progetto di blockchain a livello Europeo, indica come uno dei possibili client host Hyperledger Besu [15].

BIBLIOGRAFIA

[1] T. L. Olsen e B. Tomlin, «Industry 4.0: Opportunities and Challenges for Operations».

[2] G. Wang, «SoK: Applying Blockchain Technology in Industrial Internet of Things».

[3] T. Fernandez-Carames e P. Fraga-Lamas, «A Review on the Application of Blockchain to the Next Generation of Cybersecure Industry 4.0 Smart Factories».

[4] B. Esmaeilian, J. Sarkis, K. Lewis e S. Behdad, «Blockchain for the future of sustainable supply chain management in Industry 4.0».

[5] «Hyperledger Besu Annula Report 2020,» [Online]. Available: https://www.hyperledger.org/learn/publications/hyperledger-annual-report-2020.

[6] «Poste Italiane Turns to Hyperledger Besu to Streamline Multibrand Loyalty Program,» [Online]. Available: https://www.hyperledger.org/blog/2021/05/19/poste-italiane-turns-to-hyperledger-besu-to-streamline-multibrand-loyalty-program.

[7] [Online]. Available: https://www.hyperledger.org/blog/2019/08/29/announcing-hyperledger-besu.

[8] «Ethereum Wiki,» [Online]. Available: https://eth.wiki/concepts/ethash/ethash.

[9] «L'algoritmo Clique — Consenso Proof of Authority,» [Online]. Available: https://medium.com/vivido-it/lalgoritmo-clique-consenso-proof-of-authority-1995ddbe38a7.

[10] «Another day, another consensus algorithm. Why IBFT 2.0?,» [Online]. Available: https://consensys.net/blog/news/another-day-another-consensus-algorithm-why-ibft-2-0/.

[11] R. Saltini e H.-W. David, «IBFT 2.0: A Safe and Live Variation of the IBFT Blockchain».

[12] Prometheus, «https://prometheus.io/docs/introduction/overview/,» [Online].

[13] Clique, «https://besu.hyperledger.org/en/stable/Tutorials/Permissioning/Create-Permissioned-Network/,» [Online].

[14] I. 2.0, «https://besu.hyperledger.org/en/stable/Tutorials/Private-Network/Create-IBFT-Network/,» [Online].

[15] «EBSI: European Blockchain Services Infrastructure,» [Online]. Available: https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/EBSI.

[16] «EBSI Technical Requirements,» [Online]. Available: https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/Minimum+Technical+Requirements+for+an+EBSI+v1.0+NODE+Deployment.

INDICE TABELLE

INDICE FIGURE

APPENDICE

A. Istruzione del file di genesis usato per avviare la rete:

{

"config":{

"chainId":1981,

"constantinoplefixblock": 0,

"clique":{

"blockperiodseconds":1,

"epochlength":800

}

},

"coinbase":"0x0000000000000000000000000000000000000000",

"difficulty":"0x1",

"extraData":"0x0000000000000000000000000000000000000000000000000000000000000000053775bf737326d4a394d6b174ffe60056055e03e0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000",

"gasLimit":"0xa00000",

"min-gas-price":1337,

"mixHash":"0x0000000000000000000000000000000000000000000000000000000000000000",

"nonce":"0x0",

```
"timestamp":"0x5c51a607",
"alloc": {
        "BF6020816A2842EC4C61db0697E07904470ba1f3":{
        "balance":"0x90000000000000000000000"
    },
        "10e394f07a9727b2aEbB4a03d8Cf77cea47F5C76": {
        "balance": "0x9000000000000000000"
        }

},
"number":"0x0",
"gasUsed":"0x0",

"parentHash":"0x0000000000000000000000000000000000000000000000000000000000000000"
}
```

B. Istruzioni di avvio della rete:

besu

--data-path=data

--genesis-file=../cliqueGenesis.json

--network-id 45623

--rpc-http-enabled

--rpc-http-api=ADMIN,ETH,NET,CLIQUE,WEB3

--host-allowlist=""*

--min-gas-price=0

--rpc-http-max-active-connections=50000000

--rpc-http-cors-origins="all"

--rpc-http-host=0.0.0.0

--metrics-enabled

C. Script Web3js per la creazione di account nella rete:

```
let Indirizzi = new Array (100);

const Web3 = require('web3');

const web3 = new Web3('http://192.168.1.58:8545');

for(var i = 0; i < 1001; i++){

   Address [i] = web3.eth.accounts.create(web3.utils.randomHex(32));

   console.log('Number:'+ i);

   console.log('Address:' + Address [i].address);

   console.log('PrivateKey:' + Address [i].privateKey);

}


const fs = require('fs');

const jsonContent = JSON.stringify(Address);


fs.writeFile("./Address.json", jsonContent, 'utf8', function (err) {

    if (err) {

        return console.log(err);

    }


    console.log("The file was saved!");

});
```

D. Script Web3js per l'invio di transazioni tra account nella rete:

```
let fileAdress = require("./Indirizzi.json");

function Transaction (Alice, Bob){
   var Alice, Bob;


const Web3 = require('web3');
var i = 0;
const web3 = new Web3('http://192.168.1.58:8545');
const deploy = async () => {
   console.log(
      `Attempting to make transaction from ${Alice} to ${Bob}`
   );
for(i=0; i < 1800 ;i++){
   const stringa = "test";
   const createTransaction = await web3.eth.accounts.signTransaction(
      {
         to: Bob,
         //value: web3.utils.toWei('0', 'ether'),
         data: web3.utils.toHex(stringa),
         gas: '21476',
      },


      Alice
   );
   const createReceipt = await web3.eth.sendSignedTransaction(
```

```
      createTransaction.rawTransaction,
   );
   console.log(`Transaction successful with hash: ${createReceipt.transactio
nHash}`);
}
};

deploy();
}
for (var i = 0; i < 100; i++){
    Transaction(Address[i].privateKey, Address[i+1].address);   }
```