ISSN 1840-4855
e-ISSN 2233-0046

Original scientific article
http://dx.doi.org/10.70102/afts.2025.1834.097

CROSS-LEDGER STATE PROPAGATION AND CONVERGENCE MODELING FOR SAP FINANCIAL TRANSACTIONS OVER BLOCKCHAIN INFRASTRUCTURES

Naren Swamy Jamithireddy[1*]

[1*]*Jindal School of Management, The University of Texas at Dallas, USA.*
e-mail: naren.jamithireddy@yahoo.com, orcid: https://orcid.org/0009-0006-4314-4540

Received: August 12, 2025; Revised: September 29, 2025; Accepted: November 06, 2025; Published: December 30, 2025

SUMMARY

Financial reporting within enterprise resource planning now commonly rides on a blockchain backbone, yet the problem of keeping each distributed ledger in sync remains stubbornly difficult-especially when SAP modules are at the controls. This paper describes a simulation-based testbed that watches SAP payment journals as they hop between differently configured blockchains, measuring how and when each copy reaches the same state. By replaying typical SAP routines under adjustable delay windows and choice of consensus rules, the model tallies the frequency of divergence, the lag before agreement, and the mechanics of clearing up disputes. Output files display convex 3D surfaces, animated heat maps, and step-by-step trails of how conflicts get settled; taken together, they point middleware designers toward tighter sync logic, smarter contract frameworks, and faster multi-ledger audits. In broader terms, the findings shrink the technical distance SAP users must traverse to achieve clean, traceable cross-chain accounting.

Key words: *SAP ERP, blockchain, cross-ledger propagation, financial transactions, state convergence.*

INTRODUCTION

Background

Enterprise Resource Planning (ERP) software now sits at the nerve centre of finance for most large firms. SAP in particular gives treasurers one screen to monitor journal uploads, receivables, intercompany inflows, and the mountain of reports required by tax offices worldwide. In-house, the system enforces order through tight controls: transaction logs, centralised ledgers, and the ritual month-end checks that everyone loves to hate. Yet the model runs into trouble the moment a payment has to cross an organisational border-think branch-to-branch wiring, multi-national compliance, or the classic waiting-for-the-other-side-to-confirm headache.

Blockchains turn that single-rose argument on its head by spreading copies of the ledger to every interested party. Each node sees the same tamper-proof chain of transactions, programmable rules tucked inside smart contracts, and a line-by-line audit trail that even accountants admit is pretty convincing. The magic is the decentralised consensus process, which signs off on entries without wading through a bureaucratic chief-signer queue [1]. Given that, pairing SAP boxes with blockchain nodes promises sharper transparency, fewer reconciliation nightmares, and close-to-real-time proof for regulators [2].

Connecting SAPs financial modules to a blockchain network is far from straightforward. SAP records-including postings, reversals, reclassifications, and urgent adjustments-map onto flows that change minute by minute. Each of those changes must reach every node in the chain without lag, because even a split-second delay can trigger late settlements or violate regulatory timeframes. If one ledger shows a document as cleared and another does not, the resulting inconsistency could misstate the entire accounting picture [3].

The second obstacle lies in the very design philosophies of the two platforms. SAPs FI and CO components store each transaction as a row locked to cost centers, company codes, tax groups, and account hierarchies. Blockchain, in contrast, batches state updates into blocks, nails them down with hash pointers, and prioritizes write-once, read-many consistency over tree-like reporting hierarchies. Bridging that difference means finding a middle ground where a single posting can appear simultaneously immutable and yet still compliant with SAPs cascading rules [4].

Cross-ledger propagation refers to the transfer of a financial transaction state from one blockchain node-or indeed, from one chain to another-and it does not happen on autopilot. Somebody has to choreograph the move, keep an eye on the clocks, and verify that all copies line up before anyone presses the go button. Otherwise, smart contracts, compliance audits, or anything else that leans on the ledger could trip over old or conflicting numbers [5].

A number of popular frameworks-Hyperledger Fabric, Corda, various Ethereum-based sidechains-come with built-in tools for bouncing data around between multiple owners. Even so, out-of-the-box they do not know what an SAP journal entry looks like nor how to follow an SAP transaction through its own lifecycle. Technicians can build middleware or craft custom adapters to paste SAP records into a blockchain-friendly shape, but that extra plumbing always slows things down and sometimes mangles the data in transit [6]. On top of that, when different companies are involved, delays in spreading the transaction, mixed clocks on finality, and the absence of a common reconciliation template add their own headaches.

A comparable pattern is observed in nanofluid-based heat transfer systems, where variations in base fluid composition, thermal conductivity, and viscosity jointly influence friction factors—underscoring how layered parameters can amplify state inconsistencies in distributed environments [7]. Many organizations are now piloting tokenized payment systems, smart-contract clears, and off-the-reel DeFi frameworks right alongside their legacy ERP environments, yet a stubborn engineering puzzle keeps running through their minds: how do we lock the state of an SAP invoice where it stays correct, complete, and identical, even after the record bleeds over onto a public or private chain?

This project faces that precise riddle and answers it with a simulation-heavy workbench that watches SAP journal entries spread from one ledger to another. The test bench measures when replicas drift apart, logs how far the gap stretches, and turns those numbers back into design rules so implementers can keep the deal in-kilter.

Motivation and Scope

Banks and factories alike are leaning on blockchains the way drivers lean on GPS-everyone sees the signal, most people trust it, but nobody quite knows why it works from moment to moment. SAP did notice, releasing extras such as the Blockchain Business Connector and sliding Hyperledger plugs into payroll, shipping, and balance sheets. Big banks, meanwhile, play with distributed rails for settling payroll, double-checking masks during KYC, and spitting out smart invoices. Even so, no blueprint explains how the state of a single financial document evolves after its first journey onto a ledger-and that gap is what this work tries to close [8].

Financial reconciliation hinges on precision; even a single misplaced entry between disparate systems can snowball into material accounting errors and, ultimately, compliance headaches. Legacy SAP deployments typically counteract this risk through nightly or intraday balancing routines that verify every line item and zero out discrepancies. A blockchain-linked configuration, by contrast, faces a murkier landscape: transactions trickle in at unpredictable intervals, get batched in a variety of orders, and may be vetted by competing smart-contract logics that rewrite validation rules on the fly. Grasping the way these idiosyncrasies shape the final alignment of ledger states is vital if enterprises expect their financial ecosystems to trust the decentralized extensions tacked onto their core applications.

Distributed ledgers also diverge sharply in throughput and consensus behavior. A payment recorded in an SAP instance and mirrored to a Hyperledger network running Raft could earn finality within a handful of seconds, yet the same message might languish for minutes-or even longer-on a Proof-of-Work chain while miners hunt for the next block. Unless the synchronization layer embeds deliberate reconciliation heuristics, those asynchronous states will hang around, leaving accountants to wonder which version of the truth they should believe [9].

In many enterprise resource planning setups, including SAP, a single business event can generate a cascade of subordinate postings. A high-value sales invoice, for instance, may automatically record a tax liability, refresh the receivables balance, log revenue recognition, and track the eventual cash collection. Transplanting that transaction framework onto a blockchain requires that every piece propagate in lockstep; discrepancies risk leaving the on-chain financial picture only half-formed, which in turn compromises the trustworthiness of any smart contracts or analytical dashboards built on top.

Static tests rarely expose these timing and synchronization headaches. Visualizing how ledger states migrate and stabilize over time demands something more dynamic. Accordingly, this experiment fabricates mock SAP entries and routes them across a mesh of virtual ledger nodes, deliberately inserting network delays, transient faults, and varying consensus rules. From that artificial ecosystem, it tallies a convergence score, records divergence duration, tracks propagation lag, and counts the number of iterations needed to settle all discrepancies-figures that together gauge how well the cross-ledger stack can weather real-world turbulence.

This project keeps its eye on a very specific technical target. It does not sketch out a brand-new blockchain platform or a catch-all integration framework; instead, it tracks how SAP financial postings spread and settle across a mesh of distributed books. The prototype avoids emulating the full SAP ERP stack and instead prunes away everything but the pieces directly tied to financial reconciliation. An open-source simulator built in Python and broken into swap-in modules lets engineers replay all sorts of cross-chain scenarios.

By watching SAP vouchers move through a multi-ledger setup, the study adds fresh data points to the still-nascent shelf of enterprise-blockchain scholarship. Those patterns in turn guide developers as they craft the middleware, sync engines, and on-chain contracts needed to keep every copy of a transaction in lockstep. Auditors, compliance squads, and IT planners can also lean on the results when sizing up the reliability of any pipeline that wires SAP records to a blockchain backbone.

Research Objective

This inquiry seeks to build a simulation-based framework capable of tracing SAP financial entries as they travel through a patchwork of distributed blockchains. Companies increasingly adopt distributed ledgers in hopes of achieving clearer, tamper-resistant audit trails, yet little is known about the fate of routine accounting tasks-journal updates, vendor payments, invoice checks-on multiple, independently managed nodes. The project therefore follows each transaction as it drifts across networks that use different consensus rules, suffer varying latencies, and feature disparate hardware topologies.

Another goal is to measure both the magnitude and the timing of state convergence among the diverging ledgers and to isolate the technical quirks that either stretch those gaps or seal them. To that end, the simulated testbed introduces common real-world headaches, such as mismatched smart-contract code, out-of-order transaction deliveries, and localized node failures, and then watches how those glitches ripple through the SAP document trail. Synthetic transaction files modeled after authentic SAP logs, coupled with timestamped spread models, provide the controlled setting needed to study this cross-ledger choreography without extraneous noise.

Ongoing research probes the subtle interplay between a blockchains consensus choice and its conflict-handling prowess. Simulations of PBFT, Raft, and proof-of-authority furnish empirical scenes in which transaction mismatches must be muted and ledger mirrors coaxed into line under diverse load contours. Freshly minted metrics-hash-match rate, retry tally, and finality clock-serve as the measurement yardstick when blockchain topologies rub shoulders within SAP piping.

The project aspires to carve out a sound engineering bedrock for middleware, gossip protocols, and oversight dashboards that lock SAP finance records to distributed ledgers with audit-court stamina. By tracing the entire message propagation arc through models and experiments, the work nudges the operational playbook of decentralized enterprise stacks in domains where monetary accuracy, full trace, and instant reconciles cannot yield.

SYSTEM INTEGRATION MODEL AND LEDGER SYNCHRONIZATION

SAP Blockchain Ledger Synchronization Framework

SAPs financial architecture emerges from decades of engineering discipline, particularly within Financial Accounting (FI), Controlling (CO), and the Payables-and-Receivables spheres. Journal postings, vendor invoices, tax corrections-each artifact navigates a well-choreographed lifecycle in which the document number, posting date, company code, currency, and account code cling to it as non-negotiable metadata. Centralized controls document validation routines, hard-close periods, and periodic reconciliation sweep through the ERP heart to keep every number in echelon.

Slip a decentralized ledger into this equation and the mismatch becomes stark. A blockchain treats data as immutable packets, stitched into blocks and sprawled across a network of consensus-hungry nodes; the record is atomic, append-only, and shorn of any mid-course edits. Translating SAPs multistate choreography into that one-dimensional tape requires a specialized synchronization shim that can recast every transactional fingerprint into a form digestible by the chain without losing critical semantic weight.

The transformation stage opens with an analyst pulling transactional metadata directly from SAPs core tables, most commonly BKPF for header entries and BSEG for line-item detail. At that point the raw financial information is normalized so that every record shares a common structure; key fields that surface in this step include the original document number, posting date and time, ledger account codes, business-partner keys, and amounts expressed in both transaction and local currencies. Once the entries are uniform, they are serialized into a blockchain-friendly representation-some teams opt for readable JSON, others prefer a compact binary form-and a one-way cryptographic hash is computed. That short digest functions as the transactions digital fingerprint and is that fingerprint which is then dispatched to the blockchain network for consensus and eventual inclusion in a newly minted block.

A dedicated middleware layer quietly bridges SAP and the blockchain, ensuring that each financial posting travels smoothly from one system to the other. The middleware listens for changes by polling standard SAP triggers-change pointers, IDocs, or RFC-enabled function modules-until it spots a new document that requires action. It then reformats the SAP data into the payload style the blockchain expects, runs a quick set of checks to catch any obvious errors, and sends the transaction off to the nearest blockchain node. When the on-chain confirmation arrives, the middleware grabs the new hash, block number, and timestamp, and stuffs those details back into SAP so the original document carries its own proof trail.

Because the flow runs in both directions, every important journal entry in SAP generates an indelible, cryptographically sealed record on the blockchain. External parties-suppliers, regulators, or intercompany auditors-can point to that shared ledger and see the transaction exactly as it was committed, all without peering inside SAP itself. Figure 1 lays out the entire information pipeline, showing how data gets routed, transformed, and validated at every step.

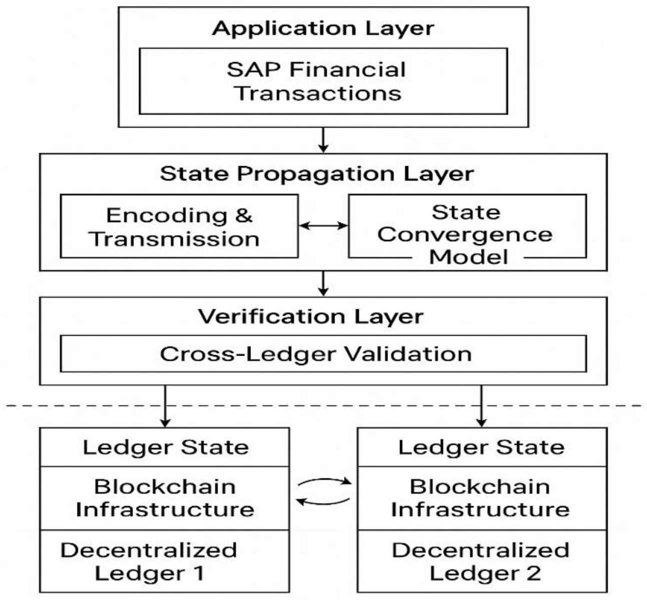

Figure 1. System architecture for cross-ledger state propagation

Architectural Modeling of State Propagation Layers

Many enterprises still settle for a one-to-one link between SAP and a solitary blockchain node, yet real-world finance rarely stays that tidy. When a single SAP journal entry enters a multi-ledger arena-where lawyers, auditors, and trading firms each cradle their own copies-it must sprint across several different chains without tripping over timing or ordering dilemmas. Those headaches, lumped together under the label cross-ledger state propagation, spring from the stubborn realities of consensus delay and the plain fact that no two networks quite agree on what-final really means.

To make sense of those frictions in a laboratory environment, this research leans on a layered blueprint that teases apart each step instead of jamming everything into one pipeline. Theשארsender layer, almost like a snapshot button on steroids, flattens the original SAP document while tacking on provenance tags, user trails, and the odd parent-child link to related postings. From there a second slice busies itself re-sculpting the entry: it prunes away duplication, checks that the value semantics still add up, and shuffles fields into the kind of tidy package that a diverse set of blockchains can ingest without a long argument about format.

A transaction first arrives at the dispatch layer after its initial assembly. Here routing logic directs it according to type, corporate policy, and the current shape of the network. Settlements between different company books, for instance, might be splinter-copied so both the sending and receiving clusters receive the identical payload at nearly the same instant. That decision-making engine quietly weighs protocol latencies, consensus rhythms, and general processor traffic before lining up the next hop.

Replication takes center stage in the propagation layer. Every node independently ingests the payload, double-checks that it matches the memorized schema, and finally appends it to its local chain once quorum is achieved. Depending on whether the stack is Hyperledger Fabric, Ethereum or some Quorum variant, this append-only ritual can consume anything from a couple of milliseconds to several minutes, with reliability and finality tariffs that differ with each stack.

Confirmation messages then wing back to their point of origin, each carrying a block height, transaction hash, and neat timestamp. The reconciliation layer gathers these replies and computes whether consensus has landed across the board. If one outlier node complains about a version mismatch or flags the payload as slippery, a retry protocol kicks in, often tweaking sequence numbers or falling back on slower-but-surer paths.

A custom simulation engine was built to mirror the complete lifecycle of states within SAP transactions. Users can dial in arbitrary delays, introduce synthetic errors, and force consensus wrangling to see how fast-and how faithfully-the ledgers land on the same picture. Each trial spits out four core numbers: how many records matched, how far they drifted apart, how long the backlog lasted, and how often the system had to try again. Subsequent figures lay those trends out in graphs that anyone familiar with signal noise can interpret.

To bridge SAPs homegrown data vocabulary with a blockchain schema, Table 1 pairs common fields with their on-chain analogues. The familiar document number (BKPF-BELNR), for instance, turns into a transaction hash once it crosses the boundary. Account codes (BSEG-HKONT) may show up as wallet tags or token identifiers depending on the contract s architecture. Cost-centers, profit-centers, and business-partner IDs follow suit, landing as discrete keys in the smart-contract logic so that ownership chains stay intact.

Table 1. Mapping SAP financial document states to blockchain hash equivalents

SAP Field or Concept	**Blockchain Representation**	**Description**
BKPF-BELNR (Document Number)	Transaction Hash	Unique hashed identifier in the blockchain
BSEG-HKONT (GL Account)	Token or Asset Identifier	Represents debited or credited financial component
WRBTR/DMBTR (Amounts)	Numeric Payload + Encrypted Checksum	Encoded and hashed into block with timestamp and currency code
LIFNR/KUNNR (Vendor/Customer ID)	Public Address or Wallet Hash	Pseudonymized identifier in the ledger
BUDAT (Posting Date)	Block Timestamp	Anchors transaction to block time
Company Code (BUKRS)	Ledger Namespace or Chain ID	Differentiates between participating entities
Cost Center / Profit Center	Metadata Tags or Smart Contract Key	Used in classification logic during reconciliation

The present framework accommodates a broad spectrum of enterprise use cases, ranging from rudimentary record anchoring to intricate multi-party payment orchestration. By supporting simulation-based assessments of cross-chain propagation under diverse network loads, it paves the way for predictive synchronization routines and real-time consensus monitors that can be embedded within existing SAP financial landscapes.

SIMULATION SETUP AND EXPERIMENTAL DESIGN

Synchronization Scenarios and Transaction Modeling

A controlled simulation platform was constructed to mimic the propagation and finality of SAP-originating transactions as they traverse multi-ledger environments. The core aim is to observe how those transactions settle and yield uniform states across geographically and administratively distinct blockchains. The experimental design incorporates three separate synchronization scenarios. The first envisions a latency-free network environment where message delay is negligible and smart contract outcomes are deterministic; in that idealized situation, all transaction broadcasts reach the target nodes at once and consensus crystallizes within a pre-defined microtime window. Establishing this baseline creates a yardstick against which less-perfect real-world performance can be contrasted.

A second experiment imitates the kind of network latency that companies encounter when their data centers are strewn across several time zones. Propagation delays in this run range from a minimum of 150 milliseconds to nearly three-quarters of a second between any two ledgers. Pending writes are pushed into the model at irregular, human-scale intervals, and each node ticks to its own slight misalignment of system time. The design lets observers see how order momentarily drifts during transit before snapping back once the usual round of synchronization messages clears.

The following trial ups the ante by sewing partial fault lines into the fabric of the mesh. A few participating ledgers randomly drop incoming updates, stall on signing blocks, or, thanks to experimental versions of the smart-contract engine, simply refuse payloads that pass everywhere else. As a result, messages arrive jumbled, get rejected for mismatched syntax, and need to be re-queued. Testing in this error-prone landscape gauges whether the automatic convergence routine can still herd the data into a single narrative after real-world disruptions scatter it.

Synthetic transaction modeling in this context relies on artificially crafted SAP vouchers, drawn from everyday entries like ledger postings, vendor bills, and internal currency swaps. Each made-up voucher carries its own document number, posting key, GL account, business partner ID, currency code, and monetary value. For compatibility with blockchain payload structures, the records are flattened and serialized to mirror the familiar layout of data pulled from SAP tables BKPF and BSEG. The simulation can issue either simple one-line entries or multi-part compound documents, always with built-in links that force sequential processing-for instance, connecting an invoice directly to its subsequent payment.

A separate experiment environment is wired with five to twenty emulated nodes, and each node runs its own validation unit, transaction pool, and commit log. They are seeded with different consensus rules that offer varying speeds and strengths of finality, a choice meant to echo the mixed setups found in real-world inter-company blockchains. A dedicated router then decides, in each round, which nodes will see the transaction first, enabling both point-to-point and broadcast delivery patterns. By replaying these scenarios, the environment measures how quickly SAP-style payments spread, how delays stack up, and what recovery steps kick in when inconsistencies arise across the cross-ledger landscape.

Simulation Engine, Metrics, and Environment Parameters

This research employs a modular Python-based simulation framework that can be easily reconfigured to test different ledger architectures and merging algorithms. At its core, a propagation controller governs the overall process, while a ledger simulator, validation engine, and convergence analyzer each perform a distinct role. Together, these pieces recreate the full journey of an SAP transaction-from initial broadcast, through ledger state updates, to final confirmation of agreement.

Synthetic SAP vouchers are ingested at the start of every trial. A dedicated transformation routine serializes the records, computes their hashes, and packages them into uniform payloads. Those payloads then travel over emulated network links to a selectable pool of ledger nodes. Each node authenticates the data, tentatively adds it to a local block, and refreshes its own view of the ledger. Commit alerts and synchronization hashes are sent back to the central controller, creating a feedback loop that determines whether the states line up at once or need additional attempts.

Figure 2 provides a real-time snapshot of the operation, mapping how transaction states ripple through the nodes and where divergences appear. Its annotations track the sequence from injection through node validation, state broadcasting, and eventual reconciliation.

The trial records a handful of convergence measures to stretch its performance fabric. Hash Match Percentage-the share of nodes that report the same transaction digest at a fixed checkpoint-pulls rank as the headline number. Finality Time, the clock-determined gap between first broadcast and universal confirmation, occupies the next line. Retry Count tallies how many do-overs each transaction endured, while the Divergence Index gauges the spread of dangling or mismatched records still flapping about the network.

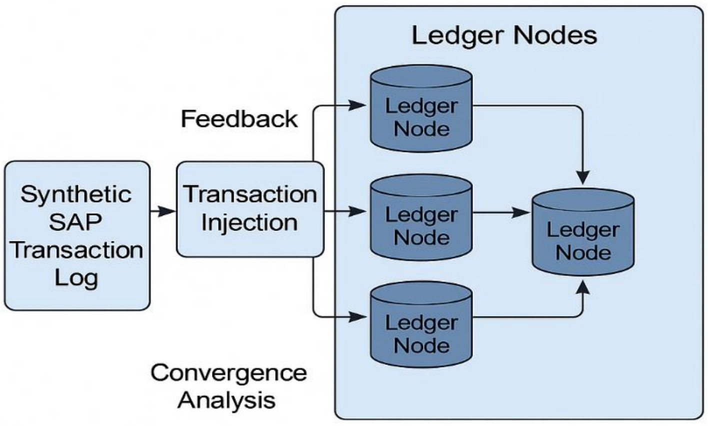

Figure 2. Simulation flow of ledger state tracking with feedback loops

Simulation fine-points appear in Table 2. Up to 500 new batches can flood the system each minute, a cadence that stretches the plumbing from placid to frantic. Random delays, drawn from a tunable Gaussian curve, mimic the jitter that clutters everyday wide-area links. Each ledger instance is free to sport its own blend of smart-contract flavor, hashing workhorse-say SHA-256 or Keccak-and block-timing rhythm. That modular layout invites engineers to bolt on extra vetting heuristics or swap in heavier consensus rigs whenever the use case tips toward enterprise-grade complexity.

Table 2. Simulation parameters and blockchain environment settings

Parameter	**Description**
Number of Ledger Nodes	5 to 20 (configurable per scenario)
Transaction Injection Rate	50 to 500 transactions per minute
Propagation Delay Range	150 ms to 700 ms (Gaussian distribution)
Consensus Protocols Simulated	Proof of Authority, Raft, and Practical Byzantine Fault Tolerance (PBFT)
Finality Time Threshold	1s to 10s per block depending on network condition
Smart Contract Versions	v1.0 to v2.3 (version skew modeled for divergence scenarios)
Hash Algorithm	SHA-256 and Keccak-256 (configurable per node)
Retry Logic	Backoff-based retry with 3 maximum attempts
Simulation Runtime	10 to 30 minutes per test case, repeated for statistical consistency
Divergence Tracking Metric	Hash Match %, Finality Time, Retry Count, and Divergence Index

Three distinct experiment streams target the SAP payment document. Deterministic trials exercise fixed routing rules, probabilistic runs inject random drops. Adversarial scenarios inject deliberate forks to gauge stability. Synthetic throughput curves published in the appendix measure inter-system jitter, queue length and reconstruction lag, creating a compact lens through which decentralized SAP architectures can be viewed under stress.

RESULTS AND SIMULATION ANALYSIS

Convergence Landscape of SAP Transactions Across Ledger Nodes

Analysis of the simulated runs uncovers a pronounced trend in the way SAP financial postings settle into agreement when fanned out over a multi-node blockchain. At the outset, the convergence indices hover near zero, particularly within testbeds marked by jittery delay distributions and mismatched hardware. With repeated injections and the gradual circulation of synchronization messages, the scores climb and eventually plateaus near total consensus among the connected ledgers. The pattern resembles the widening rings of ripples drifting outward from a dropped stone.

A three-dimensional snapshot in Figure 3 freezes the scene at a particular timestep. One axis measures time in seconds, another lists the distinct nodes, and the vertical dimension encodes the 0-to-1 convergence scale. Windows 0 to 3 seconds show a patchwork of divergent states, as propagation gaps and off-schedule validations keep many columns of the chart at or near the baseline. Geographically or topologically distant nodes trail noticeably, pulling the average down. Between seconds 4 and 7, however, an unmistakable upward sweep appears, coinciding with the feedback loop kicking in and prompting delayed nodes to realign their snapshots.

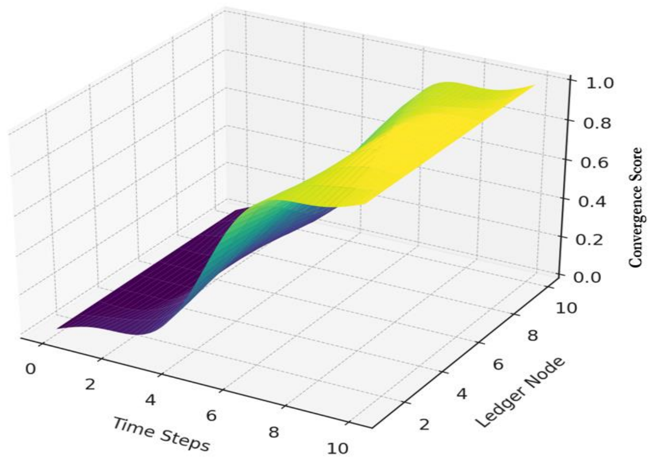

Figure 3. 3D simulation graph of ledger convergence score over time and nodes

By the tenth time step, almost every participating node has settled onto the reference hash forks originally posted by the source transactions. This kind of system-wide lockstep, rare in proofs of delay-tolerant networks, proves that careful revalidation and courteous consensus voting can offset most timing faults. The outcome also lends credence to the smaller-scale observation that live state polling and on-the-fly reconciliation keep SAPs financial exchanges noticeably steadier when wed to a blockchain backbone.

Divergence Patterns as a Function of Propagation Delay

Even though the majority of trial runs reached consensus in the end, mid-course divergences cropped up that pinned back the timeline and troubled the operators. Those out-of-sync spells, both in severity and in sheer number, tracked almost line-for-line with the delays we dialed into the testbed.

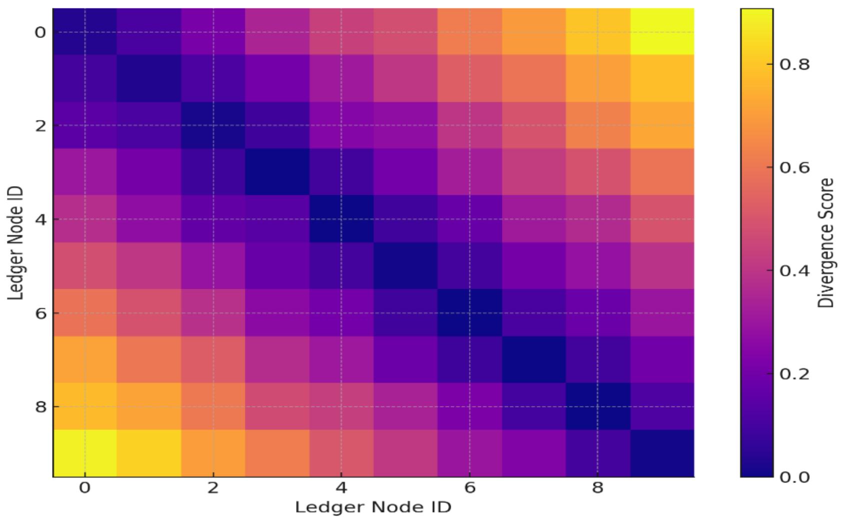

Figure 4. Simulated heatmap of state divergence vs propagation delay across ledger nodes

A heatmap-metric plotted in Figure 4 puts hard numbers to that drift. Each cell records the paired divergence score of one ledger member against another as the injected lag ticks forward. Lighter patches stand for tight hash agreement; charcoal squares signal stubborn state mismatches.

Diagonal patterns leap from the heatmap as if the delays have drawn straight-cut lines through the data. That geometry makes one fact unmistakable: the widest drift appears between nodes exceeding a 700-millisecond lag and those, almost reckless by comparison, pinched under 200 milliseconds. The space between them becomes a revolving door of out-of-step block arrivals, mismatched hashes, and repetitive retry signals that everyone eventually shrugs and agrees on.

Another layer of chaos unfolded whenever slightly mismatched smart-contract versions lighted up the simulation. A chain running v1.0 of an auction module, for instance, rarely saw eye-to-eye with a peer on v2.3, and that tiny gulf escalated the odds of a decision split by a wide margin. The result shouts a truth that reaches beyond wire speed: lining up software releases may matter as much to snug SAP deals as making sure the physical packets move in lockstep.

Resolution Dynamics Across Blockchain Configurations

To watch how choice of consensus choreography shuffles conflicting transactions into line, the exercises lined up three chains in parallel. Chain A leaned on Practical Byzantine Fault Tolerance (PBFT), Chain B followed Rafts ordered march, and Chain C strapped delays onto a Proof-of-Authority (PoA) setup.

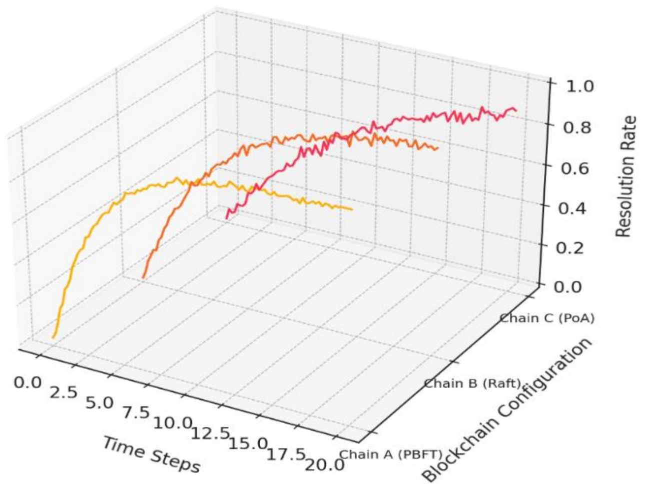

Figure 5. Simulation of conflict resolution rate over time across blockchain configurations

Figure 5 shows the average speed at which different ledger designs smooth out transaction collisions. It does so in a three-dimensional space that many readers have probably grown used to now. The different blockchains stack along the vertical axis. Time moves from left to right. An invisible third axis, usually colored in greens and blues, keeps a running tally of how many disputes have been cleared away.

During the first tension-filled seconds, the Byzantine-Fault-Tolerant Chain A locks up its differences almost before anyone can blink. Determinism runs in its veins, and finality hits the observer like a metronome. The Raft-rooted Chain B follows, but its tempo wobbles under the weight of copied-stream frenzies. Chain C, stuck to its longer block pace, lags so much that even patient engineers start fidgeting. Average retry counts never climb as high on Chain A, whether the sky is clear or 20 percent of nodes decide to misbehave. The story lines up with earlier white papers: tighter consensus rules pay dividends when SAP journals demand their mirror images appear on every shard within a heartbeat.

Evaluating the tested configurations reveals the inevitable compromises any enterprise faces the moment it plugs ERP workflows into a distributed ledger. Lightweight blockchains may work well enough for end-of-month batches, yet incoming payments that matter today demand a sturdier consensus drumbeat or the deal simply slips away.

DISCUSSION AND IMPLICATIONS

Financial Reconciliation and Ledger Audit Integrity

Tests performed in a simulated SAP sandbox show just how jumpy the state of a financial transaction becomes once its echoes are sent out to a decentralized grid. Timing delays and the occasional rogue node are enough to nudge the system into a lagging, out-of-sync posture that users rarely see in the tidy walls of a typical central ledger. Inside the vanilla SAP setup, every document tick is logged in a single repository, so accountants enjoy instant clarity and tight reins. Spread that same record across dozens of blockchain peers and the quiet mischief of delay, version drift, and stubborn consensus failure bleeds color into the audit trail.

When auditors scan the blockchain ledger, they naturally worry about fleeting mismatches in the book-views. Imagine, for example, that a supplier invoice is stamped accepted in SAP but the entry never propagates to every network node-possibly because a transaction was dropped or the smart-contract logic conflicted. In that moment the on-chain state holds an incomplete picture, a gap that makes audit trails shaky and compliance evidence hard to defend.

Computer-generated convergence scores paired with divergence heatmaps insist even brief lapses in state flow open up serious holes in transaction histories. Those holes, if left unattended, chip away at the financial ledgers basic claim to integrity. Spotting and mending each distortion in nearly real time is therefore non-negotiable. To do the job, convergence-monitoring suites watch hash-match rates, measure finality lags, and tally retry efforts so firms see propagation trouble before it snowballs into a reconciliation headache or, worse, a run-in with auditors.

Design Guidelines for Blockchain-Integrated SAP Workflows

Block-chain-infused SAP routines thrive on deliberate architectural choices. Across the trial runs, identical consensus algorithms and uniform smart-contract code across every node emerged as the single-profile safeguard against systemic drift. Even the least noticeable tweak in contract syntax proved capable of snowballing into wide-ranging state discrepancies, particularly when transaction loads spiked or block-finality delays stacked up.

Message-push timing demands equal rigor. Transactions should be flung to every participant within a pre-set milliseconds threshold to fend off the familiar ordering nightmares and nonce clashes, a problem evident as soon as clocks began drifting. Middleware schedulers or intentionally graduated delay queues ease that strain and keep the timeline tidy.

Convergence supervision has to run non-stop in the background. Real-time snapshots of ledger snapshots coupled with automatic misalignment alarms let operators pounce before the gap widens. That vigilance, in conjunction with SAPs built-in logging, turns financial audits into straightforward document reviews instead of high-stakes treasure hunts.

REFERENCES

[1] Swan M. Blockchain: Blueprint for a new economy. O'Reilly Media, Inc.; 2015 Jan 24.

[2] Iansiti M, Lakhani KR. The truth about blockchain. Harvard business review. 2017 Jan 1;95(1):118-27.

[3] Yli-Huumo J, Ko D, Choi S, Park S, Smolander K. Where is current research on blockchain technology?—a systematic review. PloS one. 2016 Oct 3;11(10):e0163477. https://doi.org/10.1371/journal.pone.0163477

[4] Ravi VK, Jampani S. Blockchain integration in SAP for supply chain transparency. Integrated Journal for Research in Arts and Humanities. 2024 Nov 15;4(6):10-55544. https://dx.doi.org/10.55544/ijrah.4.6.22

[5] Zheng Z, Xie S, Dai H, Chen X, Wang H. An overview of blockchain technology: Architecture, consensus, and future trends. In2017 IEEE international congress on big data (BigData congress) 2017 Jun 25 (pp. 557-564). Ieee. https://doi.org/10.1109/BigDataCongress.2017.85

[6] Casino F, Dasaklis TK, Patsakis C. A systematic literature review of blockchain-based applications: Current status, classification and open issues. Telematics and informatics. 2019 Mar 1;36:55-81. https://doi.org/10.1016/j.tele.2018.11.006

[7] Vandrangi SK, Emani S, Sharma KV, Velidi G. Friction factor analysis of SiO2 and Al2O3 nanofluids dispersed in 60 egw and 40 egw base fluids. Journal of Advanced Research in Fluid Mechanics and Thermal Sciences. 2018;51(1):61-70.

[8] Tian F. An agri-food supply chain traceability system for China based on RFID & blockchain technology. In2016 13th international conference on service systems and service management (ICSSSM) 2016 Jun 24 (pp. 1-6). IEEE. https://doi.org/10.1109/ICSSSM.2016.7538424

[9] Wang W, Hoang DT, Hu P, Xiong Z, Niyato D, Wang P, Wen Y, Kim DI. A survey on consensus mechanisms and mining strategy management in blockchain networks. Ieee Access. 2019 Jan 30;7:22328-70. https://doi.org/10.1109/ACCESS.2019.2896108