Математические структуры и моделирование
2026. № 1 (77). С. 78–87

УДК 510.51
DOI 10.24147/2222-8772.2026.1.78-87

ОКОННАЯ МОДЕЛЬ И РЕКУРРЕНТНЫЙ АЛГОРИТМ АГРЕГИРОВАНИЯ ТРАНЗАКЦИЙ В РАСПРЕДЕЛЁННОЙ БИЛЛИНГОВОЙ ПОДСИСТЕМЕ

И.Ф. Горкун
аспирант, e-mail: iliagorkun@yandex.ru

Омский государственный университет им. Ф.М. Достоевского, Омск, Россия

Аннотация. В работе предложена новая архитектура биллинговой подсистемы, основанная на оконной модели агрегирования транзакций и рекуррентном алгоритме обновления состояний. Проведён анализ исходного линейного алгоритма и показано, что при росте числа пользователей и транзакций он не удовлетворяет требованиям по срокам обработки и отказоустойчивости. Для устранения этих ограничений разработана формальная модель скользящего окна, описывающая поток транзакций и инкрементальное накопление агрегатов с постоянной вычислительной сложностью на событие. На основе этой модели построен двухфазный конечный автомат, реализующий транзакционность и идемпотентность в распределённой среде.

Ключевые слова: рекуррентный алгоритм, идемпотентность, конечный автомат, отказоустойчивость, горизонтальное масштабирование, распределённые системы, потоковая обработка данных.

Введение

Современные финансовые платформы неизбежно включают в свой состав компоненту биллинга, отвечающую за корректный расчёт, оформление и проведение платежей. От надёжности и точности работы этой подсистемы напрямую зависят как финансовые показатели предприятия, так и его репутационные риски. Сбой или некорректная работа биллинга может привести не только к существенным экономическим потерям, но и к юридическим последствиям, связанным с нарушением договорных и регуляторных обязательств.

В связи с этим к биллинговым системам предъявляются повышенные требования по качеству инженерной реализации, стабильности, отказоустойчивости и масштабируемости. Они должны гарантировать непрерывность обработки транзакций даже при частичных отказах инфраструктуры, а также обеспечивать соответствие бизнес-требованиям, связанным с ростом объёма операций и усложнением финансовых процессов.

Целью данного проекта являлась оптимизация существующего биллингового компонента, который столкнулся с проблемами стабильности и производительности при возросшей нагрузке. Основное внимание уделялось анализу текущих архи-

тектурных ограничений, повышению отказоустойчивости и переработке структуры системы в соответствии с современными принципами распределённых вычислений.

1. Анализ текущего алгоритма и требований

Биллинговая подсистема выполняет полный цикл расчётов для каждого пользователя в рамках банковского дня. Каждые сутки она собирает все транзакции, относящиеся к конкретному пользователю, агрегирует их по установленным правилам и формирует итоговый платёж. Этот процесс повторяется ежедневно и включает несколько стадий: выборку релевантных пользователей, обработку и суммирование транзакций, а затем выпуск платёжных документов в учётную систему. От корректности и своевременности выполнения этого цикла зависят точность расчётов и соблюдение финансовых обязательств.

Для оценки производительности и устойчивости системы необходимо формализовать её ключевые требования и ограничения. В данном разделе рассматривается исходный алгоритм работы биллинга, его временная модель и архитектурные характеристики. Особое внимание уделяется трём аспектам: соблюдению сроков завершения расчётного окна, масштабируемости при росте числа пользователей и транзакций, и устойчивости к отказам инфраструктуры.

1.1. Формализация требований

Определение 1 (Окно обработки). Пусть d — расчётный банковский день, для которого производится агрегирование транзакций и формирование платежей. Для этого дня определяются временные границы окна обработки:

$$s_d = 21{:}00\ \mathrm{UTC}\ \ d-1, \quad e_d = 03{:}30\ \mathrm{UTC}\ \ d, \quad \Delta_{win} = e_d - s_d = 6{,}5\ .$$

Определение 2. Обозначим через $T_{\text{complete}}(u, d)$ момент времени, когда для пользователя u сформирован платёж в рамках расчётного окна дня d.

Требование R1 (срок). Для каждого пользователя u и дня d должно выполняться неравенство

$$T_{\text{complete}}(u, d) \leqslant e_d,$$

то есть процесс обработки данных и создания платежа должен завершаться до конца окна e_d.

Требование R2 (масштаб). Пусть U_d — число пользователей к выставлению счетов, N_d — число транзакций за день. Для всех U_d и N_d, не превышающих верхние границы $U_{\max}$ и $N_{\max}$, система должна гарантировать корректное завершение обработки в пределах допустимого временного окна.

Требование R3 (отказоустойчивость). В кластере из M машин при одновременном выходе из строя не более $f < M$ узлов система должна обеспечивать продолжение работы и завершение обработки всех данных в пределах установленного временного окна.

SLO. Вероятностная гарантия завершения окна:

$$\mathbb{P}[T_{\text{total}}(d) \leqslant \Delta_{win}] \geqslant 1 - \varepsilon, \quad \varepsilon \leqslant 10^{-3}.$$

1.2. Изначальная архитектура

Определение 3 (Шаги исходного алгоритма). Процесс состоит из трёх стадий:

1. S1: выборка релевантных пользователей – выполняются сетевые и базовые запросы для определения активных аккаунтов;
2. S2: агрегация транзакций по пользователю – выполняются последовательные обращения к внешним системам и БД, требующие наибольших вычислительных и сетевых ресурсов;
3. S3: выпуск платежа – формирование итоговых платёжных сущностей и фиксация результатов с практически константной вычислительной сложностью на пользователя.

Модель сложности. Пусть U — число пользователей, а N – число транзакций. Введём аппроксимацию времени выполнения для каждой стадии в виде линейной модели:

$$T_1(U) = \alpha_1 U + \beta_1, \quad T_2(N) = \alpha_2 N + \beta_2, \quad T_3(U) = \alpha_3 U + \beta_3,$$

где $\alpha_i > 0$ — коэффициенты, характеризующие асимптотическую стоимость обработки одной единицы данных (пользователя или транзакции) на соответствующем этапе, а $\beta_i \geqslant 0$ — константные накладные затраты, не зависящие от размера входных данных (инициализация, синхронизация, сетевые задержки и т. п.). Совокупное время выполнения алгоритма задаётся суммой:

$$T_{\text{base}}(U, N) = T_1(U) + T_2(N) + T_3(U).$$

Лемма 1 (Неизбежность линейного роста). При фиксированной архитектуре исходного алгоритма время выполнения T_{base} увеличивается по мере роста U и N как минимум линейно.

Доказательство. Каждая транзакция просматривается не менее одного раза на стадии S2; каждый пользователь — как минимум на стадиях S1 и S3. □

1.3. Эмпирическая калибровка

Пусть при текущем масштабе (U_0, N_0) наблюдалось:

$$T_1 \approx 1{,}5\,\text{ч}, \quad T_2 \approx 3\,\text{ч}, \quad T_3 \approx 10\text{–}15\,\text{мин}.$$

Тогда $T_{\text{base}}(U_0, N_0) \approx 5\,\text{ч}$.

Следствие 1 (Порог нарушения R1). Существует порог (U^*, N^*), такой что для $U \geqslant U^*$ или $N \geqslant N^*$ выполняется

$$T_{\text{base}}(U, N) > \Delta_{win} - \delta_{ops},$$

где $\delta_{ops} > 0$ — операционный резерв (повторы, задержки). Тогда R1 нарушается для соответствующих сценариев.

1.4. Надёжность исходного запуска

Определение 4 (SPoF). Запуск на единственной машине задаёт одиночную точку отказа. При отказе в момент $t \in [s_d, e_d)$ процесс прерывается.

Теорема 1 (Уязвимость к отказам). В случае запуска на единственной машине ($M = 1$) вероятность нарушения требования R1 равна вероятности хотя бы одного сбоя в интервале $[s_d, e_d)$. Поскольку интенсивность отказов ненулевая, эта вероятность строго положительна.

Эскиз доказательства. Отказ единственной машины приводит к полной остановке процесса расчёта. Поскольку резервирование отсутствует, восстановление возможно только после ручного перезапуска, занимающего время, сопоставимое с длиной окна Δ_{win}, что делает выполнение требования R1 недостижимым при любом сбое. □

1.5. Выводы

1. Ограничение окна Δ_{win} и линейный рост T_{base} по U, N приводят к риску систематического нарушения R1 при росте нагрузки.
2. Наличие SPoF в исходной схеме создаёт риск нарушения R3.

2. Новый алгоритм

2.1. Оптимизация процесса агрегации

Из анализа исходного алгоритма следует, что наибольшие издержки приходятся на этап агрегации данных, который выполняется в оперативной памяти одной машины и зависит от внешних систем. Это создаёт риски потери промежуточных результатов и ограничивает масштабируемость.

Перед тем как перейти к описанию новой архитектуры, введём формальную модель, описывающую процесс накопления и обновления агрегатов в потоковой системе. Нас интересует поведение алгоритма в рамках одного окна расчёта, когда транзакции поступают во времени и последовательно изменяют состояние агрегата. Такое представление позволяет строго определить корректность обновления, формализовать инварианты и далее построить отказоустойчивый алгоритм, реализующий эти свойства в распределённой среде. Для этого опишем поток событий, структуру окон и рекуррентный способ обновления агрегата.

2.2. Формальная модель скользящего окна

Определение 1 (Поток и порядок). Пусть $(\mathbb{T}, \leqslant)$ — множество моментов времени с частичным порядком, отражающим причинно-временные зависимости событий. Рассмотрим поток транзакций:

$$X = \{x_i\}_{i \in I}, \quad i < j \Rightarrow (t(x_i), id(x_i)) <_{\text{lex}} (t(x_j), id(x_j)),$$

где $t(x) \in \mathbb{T}$ — метка времени, а $id(x)$ — уникальный идентификатор транзакции. Так как несколько транзакций могут иметь одинаковое значение $t(x)$, для обеспече-

ния детерминированной последовательности обработки используется лексикографический порядок по паре (t, id), аналогичный причинному порядку событий в распределённых системах [1].

Определение 2 (Окна). Фиксируем пользователя u. Его подмножество транзакций определяется как

$$X_u = \{x \in X : \mathrm{user}(x) = u\},$$

где $\mathrm{user} : X \to U$ — проекция, сопоставляющая каждой транзакции x идентификатор пользователя $u \in U$.

Пусть $\Delta > 0$ — фиксированная длина расчётного периода, а $\{b_k\}_{k\in\mathbb{Z}}$ — границы платёжных периодов, такие что $b_{k+1} - b_k = \Delta$. Окно для платежа k определяется как:

$$W_k = [b_k, b_{k+1}).$$

Требование «каждый платёж состоит из транзакций только своего периода» означает, что $\{W_k\}$ образует непересекающееся разбиение времени, аналогично оконным моделям, применяемым в потоковых системах обработки данных [2].

Определение 3 (Агрегат окна). Определим отображение

$$\varphi : X \to \mathbb{R}^m,$$

которое сопоставляет каждой транзакции x вектор её количественных характеристик (например, сумму, комиссию, налог и т.п.).:

Агрегат для платежа k:

$$A_k = \sum_{x\in X_u:\, t(x)\in W_k} \varphi(x) \in \mathbb{R}^m.$$

Лемма 1 (Рекуррентное обновление в пределах окна). Фиксируем пользователя u и текущее окно $W_k = [b_k, b_{k+1})$. Пусть транзакции пользователя в этом окне образуют поток

$$X_{u,k} = \{x \in X_u : t(x) \in W_k\},$$

отсортированный по возрастанию метки времени $t(x)$. Определим агрегат после обработки первых n транзакций как

$$A_n(u, k) = \sum_{i=1}^{n} \varphi(x_i), \quad x_i \in X_{u,k}.$$

Тогда для каждой новой транзакции $x_{n+1} \in X_{u,k}$ выполняется рекуррентное соотношение:

$$A_{n+1}(u, k) = A_n(u, k) + \varphi(x_{n+1}).$$

Доказательство. Следует из аддитивности суммирования и линейности отображения φ:

$$\sum_{i=1}^{n+1} \varphi(x_i) = \sum_{i=1}^{n} \varphi(x_i) + \varphi(x_{n+1}) = A_n(u, k) + \varphi(x_{n+1}). \square$$

Инварианты корректности

- *Линейность.* Для любых множеств $S, T \subseteq X_{u,k}$

$$\sum_{x \in S \cup T} \varphi(x) = \sum_{x \in S} \varphi(x) + \sum_{x \in T \setminus S} \varphi(x),$$

  что гарантирует корректность пошагового накопления.
- *Детерминированность.* Порядок $(t(x), id(x))$ задаёт однозначную последовательность обновлений.
- *Аддитивность.* При поступлении новой транзакции агрегат изменяется только на $\varphi(x)$, без пересчёта предыдущих данных.

Персистентность рекуррентных состояний. Для отказоустойчивости значения A_k и служебная версия хранятся в таблице агрегатов с ключом (userId, b_k), где userId обозначает уникальный идентификатор пользователя, соответствующий u. Это позволяет восстанавливать состояние без хранения всего окна в оперативной памяти.

2.3. Транзакционность и идемпотентность через конечный автомат

Для формализации поведения распределённого расчётного процесса введём конечный автомат, описывающий переходы состояний агрегата при поступлении событий и фиксации транзакций.

Пусть $\mathcal{M} = (S, \Sigma, \delta, s_0)$ – конечный автомат, где:

- S – множество состояний агрегата расчёта для пары (пользователь, период);
- Σ – множество входных событий: поступление транзакции, закрытие окна, генерация платежа и т. д.;
- $s_0 \in S$ – начальное состояние (пустые агрегаты, версия 0);
- δ – двухфазная функция перехода (‘BeforeTransaction‘ $\to$ ‘InTransaction‘), аналогичная классическим транзакционным моделям, описанным в [4].

Двухфазная семантика:

- *Подготовительная фаза* (‘BeforeTransaction‘):

$$\mathsf{pre} : S \times \Sigma \to \{\text{commit, noop, reject}\} \times M,$$

  где M — множество модификаторов $m : S \to S$.
- *Транзакционная фаза* (‘InTransaction‘):

$$\mathsf{tx} : (S, m, k) \mapsto S,$$

  выполняется атомарно в рамках БД-транзакции с дедупликацией по идемпотентному ключу k.

Идемпотентный ключ шага определяется как Идемпотентный ключ шага определяется как

$$k = Hash(u,\, b_k,\, \mathrm{id}(x),\, \mathrm{step}),$$

где $Hash(\cdot)$ – детерминированная хэш-функция, а параметр step обозначает тип выполняемой операции (например, `AddTransaction`, `CreatePayout`).

В хранилище поддерживается уникальный индекс по k, что обеспечивает ровно-однократность эффекта.

Функция применения события:

$$\mathrm{Apply}(s,\sigma,k)=\begin{cases} s, & \mathsf{pre}(s,\sigma)=(\text{noop},\ \_)\\ s, & \mathsf{pre}(s,\sigma)=(\text{reject},\ \_)\\ m(s), & \mathsf{pre}(s,\sigma)=(\text{commit},\ m)\ \text{и}\ \mathsf{tx}(s,m,k)\ \text{успешна.}\end{cases}$$

Гарантии:

- *Идемпотентность:* $\mathrm{Apply}(s,\sigma,k)=\mathrm{Apply}(\mathrm{Apply}(s,\sigma,k),\sigma,k)$.
- *Транзакционность:* ‘InTransaction‘ обеспечивает атомарность изменений состояния и регистрации ключа k.
- *Детерминизм:* модификатор m зависит только от (s,σ); побочные эффекты вынесены в идемпотентную фазу.
- *Линеаризуемость:* каждое состояние имеет версию $v(s)\in\mathbb{N}$; обновление версии гарантирует порядок.

2.4. Создание выплаты

Фиксируем окно $W_k=[b_k,b_{k+1})$. После завершения расчётного периода инициируется событие закрытия окна:

$$\sigma_{\mathrm{close}}(u,k):\quad t_{\mathrm{now}}\geqslant b_{k+1}.$$

Это событие обозначает момент, когда все транзакции, относящиеся к окну W_k, поступили и были обработаны в агрегатах пользователей. На этом этапе агрегаты переходят из промежуточного состояния `Accumulating` в состояние `Aggregated`.

Множество кандидатов на выплату определяется как:

$$E_k=\{(u,k):A_k(u)\text{ в состоянии }\texttt{Aggregated}\text{ и }A_k(u)\ >0\},$$

Таким образом, в E_k попадают все пары (пользователь, окно), для которых накоплены непустые результаты расчёта и агрегат готов к формированию платежа.

Каждый элемент $(u,k)\in E_k$ далее становится единицей работы для распределённого пула машин, обрабатывающего выплаты параллельно.

Распределение работ по машинам (устойчивое назначение). Пусть $h:(u,k)\to\{1,\dots,P\}$ – детерминированная хэш-функция (для балансировки), но источником истины о незавершённых работах является устойчивая таблица кандидатов E_k в БД. Назначение пары (u,k) выполняется через *аренду* с ограждающим токеном (fencing):

$$\mathrm{Lease}(u,k)=(\mathrm{owner},\ \tau_{\mathrm{exp}},\ \nu),$$

где $\nu\in\mathbb{N}$ — версия аренды.Аренда захватывается атомарно при помощи операции сравнения-и-обмена (CAS) в рамках транзакции. Если текущая запись отсутствует или срок аренды истёк ($t_{\mathrm{now}}>\tau_{\mathrm{exp}}$), машина выполняет обновление:

$$(\mathrm{owner}=s,\ \tau_{\mathrm{exp}}=t_{\mathrm{now}}+T_{\mathrm{lease}},\ \nu'=\nu+1),$$

где T_{lease} – фиксированная длительность аренды (lease duration), задающая максимальное время владения задачей одной машиной. По истечении этого интервала право обработки автоматически может быть перехвачено другой машиной.

Любая последующая операция над парой (u, k) должна включать проверку текущей версии ν (*fencing check*), что предотвращает возникновение *split-brain* ситуаций при конкурентном продлении аренды и гарантирует корректную эксклюзивность доступа.

При падении машины его аренда истекает, и другая машина может захватить работу без потери данных, так как агрегаты $A_k(u)$ и статусы хранятся в БД. Любая обработка выполняется атомарно и идемпотентна по ключу Идемпотентный ключ операции формирования выплаты определяется как

$$k_{\text{pay}} = Hash(u,\, b_k,\, \text{step} = \texttt{''createPayout''},\, v(u,k)),$$

где:

- u – идентификатор пользователя;
- b_k – начало расчётного окна (идентификатор периода);
- step – тип выполняемого шага;
- $v(u,k)$ – версия состояния агрегата (u,k), увеличивающаяся при каждом обновлении.
- $Hash(\cdot)$ – детерминированная хэш-функция.

Инварианты.

- (*Непотеря работ*) Множество кандидатов E_k хранится в БД; падение машины не удаляет и не скрывает (u,k).
- (*Единоличное выполнение*) Fencing-версия ν гарантирует, что только актуальный владелец может фиксировать изменения.
- (*Идемпотентность*) Уникальный индекс по k_{pay} обеспечивает ровно-однократный эффект шага «создать выплату».
- (*Прогресс*) При наличии хотя бы одной живой машины и конечном T_{lease} все элементы E_k будут обработаны.

Формирование платежа. Пусть g – детерминированная функция построения платёжной сущности, определяющая преобразование агрегированных данных в итоговую запись о платеже:

$$P_k(u) = g\big(A_k(u),\, \text{meta}(u,k)\big),$$

где $A_k(u)$ —– рассчитанный агрегат пользователя u за окно W_k, а $\text{meta}(u,k)$ – сопутствующие метаданные, включающие реквизиты пользователя, валюту, параметры выплат и дополнительные атрибуты расчёта. Функция g возвращает детерминированный результат – структуру платёжной записи, готовую к атомарной вставке или обновлению в базе данных. Идемпотентный ключ шага:

$$k_{pay} = \text{Hash}(u, b_k, \texttt{''createPayout''}, v), \quad pId = \text{Hash}(u, b_k, \texttt{''payoutId''}).$$

Здесь строковые литералы ”createPayout” и ”payoutId” выполняют роль семантических меток в хэш-функции, что соответствует принципам идемпотентной обработки событий и дедупликации сообщений, описанным в [3].

Двухфазное выполнение:

- **Подготовка:** вычисляем $P_k(u)$, проверяем инварианты, формируем модификатор m_{pay}.
- **Транзакция:** атомарно
  1. вставляем ключ k_{pay} в таблицу дедупликации;
  2. выполняем upsert платежа по (u, b_k) и pId;
  3. переводим агрегат (u, k) в статус Paid, инкрементируем версию.

Гарантии корректности конца окна:

- Идемпотентность: повтор шага с тем же k_{pay} не изменяет состояние.
- Атомарность: все действия выполняются в одной транзакции.
- Прогресс: при наличии хотя бы одной живой машины и конечным времени аренды все $(u, k) \in E_k$ будут обработаны.
- Детерминизм: g и pId детерминированы по (u, k).

Восстановление после сбоя. Если машина падает во время шага, то:

- либо транзакция не зафиксирована (аренда истекает, работа продолжается другой машиной);
- либо зафиксирована целиком (повтор приведёт к noop по k_{pay}).

Таким образом, сохраняются корректность и завершимость процесса.

2.5. Оценка ускорения и масштабирование

Новая схема заменяет полносканную агрегацию на инкрементальные обновления внутри окна (стоимость $O(1)$ на событие $\varphi(x)$) и распределяет работу по парам (u, k) на кластер из M машин с эффективной параллельностью $P_{\text{eff}} = M \cdot \eta$ (коэффициент утилизации $\eta \in (0, 1]$). Итого оценка времени выполнения:

$$T_{\text{new}}(U, N) \;\approx\; \frac{\alpha_1 U + \alpha_2 N + \alpha_3 U}{P_{\text{eff}}} \;+\; \delta_{\text{dist}},$$

где δ_{dist} — накладные расходы распределённой координации.

Для учёта неизбежной сериализации используем закон Амдала. Пусть $f_s \in [0, 1)$ – доля неизпараллеливаемых работ (координация, метаданные, публикации), тогда

$$T_{\text{new}}(U, N) \;\approx\; f_s \cdot T_{\text{base}}(U, N) \;+\; (1 - f_s) \cdot \frac{T_{\text{base}}(U, N)}{P_{\text{eff}}} \;+\; \delta_{\text{dist}}.$$

Численный пример (на базе эмпирики). При текущем масштабе наблюдалось:

$$T_1 \approx 1{,}5 \text{ ч}, \quad T_2 \approx 2\text{–}3 \text{ ч (возьмём } 2{,}5 \text{ ч)}, \quad T_3 \approx 0{,}2 \text{ ч},$$

т.е. $T_{\text{base}} \approx 4{,}2$ ч.

Оптимистично-линейная оценка (без жёсткого сериализующего хвоста, $\delta_{\text{dist}} \approx 3$ мин, $P_{\text{eff}} = \{6, 11, 16\}$):

$$T_{\text{new}} \in \{0{,}75,\ 0{,}43,\ 0{,}31\} \text{ ч} \quad \Rightarrow \quad \text{ускорение } S = \frac{T_{\text{base}}}{T_{\text{new}}} \in \{5{,}6,\ 9{,}7,\ 13{,}4\}.$$

Реалистичная оценка с законом Амдала ($\delta_{\text{dist}} \approx 3$ мин):

$$f_s = 0{,}10,\ P_{\text{eff}} = \{8, 11, 16\} \Rightarrow T_{\text{new}} \approx \{0{,}94,\ 0{,}81,\ 0{,}71\} \text{ ч}, S \approx \{4{,}46,\ 5{,}16,\ 5{,}95\};$$
$$f_s = 0{,}05,\ P_{\text{eff}} = \{8, 11, 16\} \Rightarrow T_{\text{new}} \approx \{0{,}76,\ 0{,}62,\ 0{,}51\} \text{ ч}, S \approx \{5{,}54,\ 6{,}74,\ 8{,}25\}.$$

Вывод

В работе представлена и обоснована новая архитектура биллинга на основе оконной модели. Была описана формальная модель окон и рекуррентных агрегатов, доказана корректность их обновления и зафиксированы инварианты. Введён конечный автомат с двухфазными переходами (*подготовка* → *транзакция*) и идемпотентными ключами. Показаны свойства идемпотентности, атомарности и детерминизма, обеспечивающие линеаризуемость по агрегату и устойчивость к повторам. Формализована процедура создания выплаты с шардированием и механизмом аренды (lease), гарантирующая прогресс при частичных отказах. Оценена вычислительная стоимость поддержки агрегатов как пропорциональная числу добавленных и удалённых элементов окна. Показано, что при увеличении числа машин пропускная способность системы растёт линейно до насыщения внешних ресурсов, что позволяет достигать целевых сроков обработки.

Литература

1. Lamport L. Time, Clocks, and the Ordering of Events in a Distributed System // *Communications of the ACM*. 1978. V. 21, No. 7. P. 558–565.
2. Carbone P., Katsifodimos A., Ewen S., Markl V., Haridi S., Tzoumas K. Apache Flink: Stream and Batch Processing in a Single Engine // *IEEE Data Engineering Bulletin*. 2015. V. 38 (4). P. 28–38.
3. Kleppmann M. Designing Data-Intensive Applications. O'Reilly Media, 2017. 616 p.
4. Gray J., Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, 1993. 1070 p.

WINDOW-BASED MODEL AND RECURRENT TRANSACTION AGGREGATION ALGORITHM IN A DISTRIBUTED BILLING SUBSYSTEM

I.F. Gorkun
Ph.D. Student, e-mail: iliagorkun@yandex.ru

Dostoevsky Omsk State University, Omsk, Russia

Abstract. A new billing subsystem architecture is proposed, based on a windowed transaction aggregation model and a recurrent state-update algorithm. The analysis of the original linear algorithm shows that, with increasing numbers of users and transactions, it fails to meet timing and reliability requirements. To overcome these limitations, a formal sliding-window model is introduced, describing transaction streams and incremental aggregation with constant per-event computational cost. On this basis, a two-phase finite-state machine is constructed to ensure transactional and idempotent processing in a distributed environment.

Keywords: recurrent algorithm, idempotence, finite-state machine, fault tolerance, horizontal scalability, distributed systems, stream data processing.

Дата поступления в редакцию: 30.10.2025