A SFTS PROPERTIES AND BOUNDS

We present the fairness properties and bounds ensured by SFTS. We first prove that SFTS provides both the share guarantee and strategy proofness. Next, we present bounds on the maximum difference in usage between any two backlogged clients and the maximum delay a client can observe, which are standard in prior work **??**.

A.1 Share Guarantee

The share guarantee is the gold standard for fair schedulers because it enables performance isolation. SFTS provides it for transactions, ensuring that every client will make progress.

Theorem 1. *SFTS provides the share guarantee (a backlogged client should receive an equal share, i.e., key usage, of one of the resources it uses, regardless of the demand of other clients.).*

Proof. Consider a system with a fixed number of clients, and all clients are backlogged. Without loss of generality, we assume that all clients are active during an interval $[t_1, t_2)$. Let us say that all clients start with the same virtual time at the beginning of the backlogged interval. By design, SFTS equalizes client virtual times, and virtual time is determined by the dominant key usage per transaction (Section 5.1). For a given client c that executes transaction $T_{c,j}$, let us say its dominant key usage is $u_{c,j,k}$. Assume that the backlogged interval starts at time 0. At time t, the virtual time of this client will advance by $V(c, t)$ +/- $u_{c,j,k}$. Let us say that client c's dominant resource at time t is k. Then, client c has used k by at least $V(c, t)$ +/- $u_{c,j,k}$. Thus, each client's virtual time increases by its dominant key usage, and this is equalized by SFTS. □

A.2 Strategy-Proofness

Strategy-proofness is a standard fairness property [36] that prevents clients from "gaming" the scheduler. SFTS prevents a client from improving its own performance at the detriment of others.

Theorem 2. *SFTS ensures strategy-proofness (a client cannot increase its dominant resource usage by lying about its demand, i.e., by adding extraneous operations and/or transactions).*

Proof. Assume for a contradiction that in a continuous backlogged interval $[t_1, t_2)$, client c_1 gets more service by lying in that interval. We consider two scenarios: (a) client c_1 lies about its demand and (b) client c_1 does not lie. We assume that client c_1 gets more service in case (a) than (b). For this to be true, there must be a point in time during $[t_1, t_2)$ when client c_1 has a transaction scheduled in (a) but not in (b). Otherwise, client c_1 would get the same usage in both cases. Let t_0 be the first time in $[t_1, t_2)$ when this happens. Let us say at t_0 that client c_1's transaction is scheduled in case (a) while client c_2's transaction is scheduled in case (b). The only way this can occur is if client c_2's dominant share is greater than client c_1's dominant share in (a) but not (b). For client c_2 to have higher dominant share in (a) than (b), client c_1 must artificially inflate client c_2's dominant share. There are two ways this can happen: client c_1 adds a dummy transaction or it adds dummy operations to an existing transaction.

First, we consider the case when a client c_1 adds an extraneous transaction $\hat{T}_{c_1,j}$ to try to increase its dominant resource usage by causing conflicts with another client c_2 transaction earlier in the schedule. Let us say that $\hat{T}_{c_1,j}$ causes an additional conflict with $T_{c_2,j'}$, which would cause client c_2's usage to increase. We consider two cases: (i) when client c_2's dominant resource changes and (ii) when client c_2's dominant resource does not change. For (i), we have client c_2's dominant resource change from k to k'. We denote the old usage on key k as $u_{c_2,j,k}$, the old usage on key k' as $b_{c_2,j,k'} + \frac{d_{c_2,j,k'}}{l_{k'}}$, and the new usage on key k' as $b_{c_2,j,k'} + \frac{\hat{d}_{c_2,j,k'}}{l_{k'}}$. Since the dominant resource changes, this means that $u_{c_2,j,k} > b_{c_2,j,k'} + \frac{d_{c_2,j,k'}}{l_{k'}}$. We have that client c_2's virtual time increases by $b_{c_2,j,k'} + \frac{\hat{d}_{c_2,j,k'}}{l_{k'}} - u_{c_2,j,k}$ as a result of $\hat{T}_{c_1,j}$. Since $u_{c_2,j,k} > b_{c_2,j,k'} + \frac{d_{c_2,j,k'}}{l_{k'}}$, this means that the most client c_2's virtual time can increase by is $\frac{\hat{d}_{c_2,j,k'}}{l_{k'}}$. As SFTS spreads the the schedule-dependent usage of client c_2 evenly across conflicting requests, client c_1's virtual time will increase by $\frac{\hat{d}_{c_2,j,k'}}{l_{k'}}$. Thus, client c_1's virtual time cannot increase less than client c_2's, and it cannot manipulate the scheduler to increase its dominant share, which is a contradiction. For (ii), if the dominant resource of client c_2 does not change, then client c_2's virtual time would also increase by $\frac{\hat{d}_{c_2,j,k'}}{l_{k'}}$ at most. By the same logic as (i), client c_1's virtual time cannot increase less than client c_2's so that client c_1's transactions are scheduled earlier.

Next, we consider the case when a client c_1 adds extraneous operation(s) to transaction $T_{c_1,j}$ to produce a new transaction $\hat{T}c_1, j$ to try to increase its dominant resource usage by causing conflicts with another client c_2's transaction earlier in the schedule. By similar logic in (i), client c_1's virtual time will increase by $\frac{\hat{d}_{c_2,j,k'}}{l_{k'}}$, at least as much as client c_2's virtual time, so it cannot increase its dominant resource usage, which is a contradiction. □

A.3 Bound Between Backlogged Clients

Next, we show how SFTS guarantees that the usage between backlogged clients is bounded over a given time interval. That is, our scheduler ensures that each client will make progress during this period of time, preventing starvation. This is crucial for real-world, multi-tenant databases: in industry, long-running transactions (e.g., for background or analytical tasks) are prone to starvation since they are often given lower priority [19]. This bound depends on Δ since this value determines how much usage can differ between clients. However, it is independent of the length of the time interval as well as the number of clients in the system.

To provide this bound, we must make assumptions about client access patterns. Specifically, we adapt the notion of *dominant-resource monotonic* [35] for the transactional setting. A client is dominant-resource monotonic if, during any of its backlogged periods, its dominant resource does not change. A client in which all transactions have the same dominant resource is trivially a dominant-resource monotonic client. We use $u^{\uparrow}_{c,r}$ to denote $\max_j u_{c,j,r}$, the (unchanging) dominant resource key usage of client c.

Theorem 3. *Consider dominant-resource monotonic clients c_1 and c_2, both backlogged during the interval $[t_1, t_2)$. Let $W_{c_1}(t_1, t_2)$ and $W_{c_2}(t_1, t_2)$ be the total key usages of clients c_1 and c_2, respectively,* *on their dominant resource during interval $[t_1, t_2)$. Then, we have:*

$$\left| \frac{W_{c_1}(t_1, t_2)}{w_{c_1}} - \frac{W_{c_2}(t_1, t_2)}{w_{c_2}} \right| < \frac{s^{\uparrow}_{c_1,d_{c_1}}}{w_{c_1}} + \frac{s^{\uparrow}_{c_2,d_{c_2}}}{w_{c_2}} + \Delta$$

where $s^{\uparrow}_{c,d_c}$ represents the maximum key usage of a transaction of client c on its dominant resource d_c.

Proof. Let $v_1 = V(t_1)$ and $v_2 = V(t_2)$. Consider a client c_1, with dominant resource d_{c_1}, that is backlogged in that entire interval of time. We prove the theorem using two cases: when $v_2 - v_1 > \frac{s^{\uparrow}_{c_1,d_{c_1}}}{w_{c_1}} + \Delta$, such that client c_1 can have at least have one transaction with start time inside the interval and when the opposite is true. For both, we establish lower and upper bounds on the amount of processing time client c_1 receives in the interval. The maximum discrepancy between these two is then used to establish the theorem.

Consider the following lower bound on the amount of processing time client c_1 receives in the interval $[t_1, t_2]$ on its dominant resource d_{c_1}. We observe that the amount of processing time that client c_1 receives on its dominant resource is at least $v_2 - S(T_{c_1,j}, d_{c_1})$, where transaction $T_{c_1,j}$ is client c_1's first transaction served in the interval $[t_1, t_2]$. This is because, by definition of V, virtual time can only progress to a greater value v_g when all transactions with maximum starting time less than v_g have been served. Since client c_1 is backlogged, the start time of each of client c_1's transactions, after the j^{th}, is equal to the finish time of the previous transaction. Thus, it received processing time on its dominant resource equal to $\sum_{i=j}^{m} s_{c_1,i,d_{c_1}}$, where $T_{c_1,m}$ is the last transaction in the window (i.e., $S(T_{c_1,m}, d_{c_1}) \in [v_1, v_2]$, $F(T_{c_1,m}, d_{c_1}) \geq v_2$). Thus, $S(T_{c_1,j}, d_{c_1})$ determines the total processing time received on d_{c_1} in the interval, assuming that client c_1 appears only once in the schedule since we seek the lower bound. By the same assumption, client c_1's previous request $T_{c_1,j-1}$ has a start time less than v_1. As client c_1 is backlogged, $S(T_{c_1,j}, d_{c_1}) \leq v_1 + \frac{s^{\uparrow}_{c_1,d_{c_1}}}{w_{c_1}}$. Thus, c_1 will receive at least $(v_2 - v_1) - \frac{s^{\uparrow}_{c_1,d_{c_1}}}{w_{c_1}}$ processing time on d_{c_1} in the interval $[t_1, t_2]$.

Consider an upper bound on the amount of processing c_1 receives on d_{c_1} in $[t_1, t_2]$. The processing time is maximized when the first transaction k starts at $S(T_{c_1,j}, d_{c_1}) = v_1$, and the last transaction $T_{c_1,m}$ starts at the end of the interval (i.e., $S(T_{c_1,m}, d_{c_1}) = v_2$). Since $T_{c_1,m}$'s processing time on d_{c_1} is at most $\frac{s^{\uparrow}_{c_1,d_{c_1}}}{w_{c_1}} + \Delta$, it can receive processing time in the virtual time interval $[v_1, v_2 + \frac{S(T_{c_1,m}, d_{c_1})}{w_{c_1}} + \Delta]$. Thus, c_1 receives at most $(v_2 - v_1) + \frac{s^{\uparrow}_{c_1,d_{c_1}}}{w_{c_1}} + \Delta$ processing time on d_{c_1} in $[t_1, t_2]$.

The maximum discrepancy of processing times received by the the backlogged clients c_1 and c_2 on their respective dominant resource in the interval $[t_1, t_2]$ is therefore $\frac{s^{\uparrow}_{c_1,d_{c_1}}}{w_{c_1}} + \frac{s^{\uparrow}_{c_2,d_{c_2}}}{w_{c_2}} + \Delta$, which proves the theorem for the first case.

Now, consider the case when $v_2 - v_1 \leq \frac{s^{\uparrow}_{c,d_c}}{w_c} + \Delta$, for any client $c \in \{i, j\}$. Then, c might not get any service in $[t_1, t_2]$. The upper bound is that c has one transaction that happens to have start time in the interval. Thus, the maximum discrepancy is $\max_{c \in \{c_1, c_2\}} \left\{ \frac{s^{\uparrow}_{c,d_c}}{w_c} \right\} + \Delta$, which proves the theorem for this case. □

While the dominant-resource monotonic assumption does not hold for all transactional workloads, client requests often center around a group of hot keys both in standard OLTP benchmarks [27] and real-world applications [18]. For instance, a viral celebrity post would get many interactions during the same day. Even if workloads are not dominant-resource monotonic, each client will eventually get its fair share under the share guarantee.

A.4 Bounded Delay

We also bound the maximum delay a client can observe before one of its requests is served under SFTS. This affects tail latencies, which are an important concern for database systems [80]. While this bound depends on the existing workload, it does not require client workloads to be dominant-resource monotonic.

Theorem 4. *Assume transaction $T_{c_i,j}$ of client c_i arrives at time t, and assume client c_i is idle at time t. Let r encompass all the resources that client c_i will access in its next transaction. Let n be the total number of clients in the system. Then, the maximum delay to start serving transaction $T_{c_i,j}$, denoted as $D(T_{c_i,j})$, is bounded above by*

$$D(T_{c_i,j}) \leq \max_{r} \left(\sum_{l=1, l \neq i}^{n} \left(s^{\uparrow}_{c_l,r} + \Delta \right) \right)$$

where $s^{\uparrow}_{c,d_c}$ represents the maximum key usage of a transaction of client c on its dominant resource d_c.

Proof. If no transaction is being processed at time t, $T_{c_i,j}$ will immediately be processed, and the theorem trivially holds. Thus, assume a transaction is currently being processed with maximum start time $v_t = \max_r v(t, r)$. Since client c_i is assumed to not be backlogged, we have $S(T_{c_i,j}) = v_t$. Since processing happens with increasing start tags, all backlogged transactions T must have $\max_r S(T, r) \geq v_t$. Since each client has strictly non-zero processing time on at least one resource (its dominant resource), at most one transaction from each client can have starting time v_t. In the worst case, each of these clients requires $s^{\uparrow}_{c_l,d_{c_l}} + \Delta$ processing time, and there are $n - 1$ other clients. Consequently, the total delay is no greater than the resource that finishes last when processing all the $n - 1$ client transactions that have virtual start times within $[v_t, v_t + \Delta]$. □

This theorem bounds the maximum delay based on Δ since this value determines how much client usages can differ. In practice, it is rare for *all* clients to conflict on the same resource in real-world transactional workloads. Thus, a given client will usually only need to wait for a short amount of time before it is guaranteed service.