Algoritmo Token Bucket: Um Estudo Aprofundado

Introdução e Contexto Histórico

O algoritmo **Token Bucket** é uma técnica fundamental no controle de tráfego em redes de computadores e na limitação de taxa em sistemas distribuídos. Ele foi formalmente introduzido no final dos anos 1970 e início dos anos 1980, em um período em que as redes de computadores estavam se expandindo rapidamente e a necessidade de controlar o uso de recursos compartilhados, como largura de banda e buffers, tornava-se crítica.

Antes do Token Bucket, técnicas simples de limitação de taxa, como o **Leaky Bucket** (balde vazante), eram amplamente utilizadas. O algoritmo Leaky Bucket, inicialmente proposto por Floyd e Jacobson em 1993, regula a saída de pacotes a uma taxa constante, funcionando como um "balde furado" que deixa passar dados a uma velocidade fixa, independentemente do pico de entrada (Floyd & Jacobson, 1993). Entretanto, essa abordagem não era flexível o suficiente para permitir ráfagas de tráfego quando a rede pudesse suportá-las, limitando o desempenho em aplicações interativas e multimídia.

O Token Bucket foi proposto para superar essas limitações, permitindo controle mais dinâmico e eficiente do tráfego, ao possibilitar que ráfagas de dados fossem enviadas até um limite, respeitando um rate médio.

Definição e Funcionamento do Token Bucket

Conceitualmente, o Token Bucket é um mecanismo de controle de fluxo baseado em tokens que "pingam" (são gerados) a uma taxa constante e são acumulados até um máximo definido pela capacidade do bucket (balde). Cada token representa permissão para enviar uma unidade de dado (por exemplo, um byte ou um pacote). Para que uma aplicação envie dados, ela precisa "gastar" tokens equivalentes à quantidade de dados enviados.

Assim, quando há pouca atividade, os tokens se acumulam no balde, permitindo o envio em rajadas (burst) quando necessário, desde que haja tokens disponíveis. Se a taxa de chegada de dados for maior que a geração de tokens, o excesso é rejeitado ou atrasado, controlando o fluxo.

Formalmente:

- **Token generation rate (r):** número de tokens adicionados por unidade de tempo.
- **Bucket capacity (b):** número máximo de tokens armazenados.
- Para enviar **n** unidades de dados, são consumidos **n** tokens.

Se não houver tokens suficientes, o envio deve esperar ou ser descartado.

Essa flexibilidade torna o Token Bucket ideal para modelar e controlar tráfego variável e intermitente.

Porque Surgiu e Limitações das Abordagens Anteriores

Conforme descrevem Jacobson et al. (1988), os sistemas de redes precisam equilibrar a garantia de qualidade de serviço (QoS) e a utilização eficiente da banda. O Leaky Bucket impõe uma taxa rígida

constante, o que impede que picos de tráfego legítimos sejam absorvidos.

O Token Bucket foi criado para resolver essa limitação, permitindo que as aplicações possam aproveitar bursts de largura de banda acumulada, mantendo o controle do uso médio de recursos. Isso é especialmente importante para aplicações multimídia e interativas, que geram tráfego irregular, onde a rigidez do Leaky Bucket era um gargalo.

Vantagens do Token Bucket

- **Flexibilidade para bursts:** Permite enviar pacotes em rajadas, desde que tokens estejam disponíveis, otimizando o uso da banda e a experiência do usuário (Stallings, 2007).
- **Controle de taxa média:** Mantém o controle sobre a taxa média, prevenindo sobrecarga da rede.
- **Simplicidade e eficiência:** Implementação simples e com baixo custo computacional.
- **Adaptabilidade:** Pode ser usado em redes, sistemas operacionais e aplicações para limitar uso de recursos variados (CPU, disco, etc.).

Desvantagens do Token Bucket

- **Complexidade em ambientes altamente dinâmicos:** Em redes muito variáveis, pode ser difícil definir parâmetros ótimos de bucket e taxa (r, b).
- **Atrasos para pacotes excedentes:** Pacotes que não encontram tokens disponíveis podem sofrer atrasos ou descartes, impactando aplicações sensíveis a latência.
- **Não garante QoS estrita:** Apenas limita o tráfego, não controla a latência ou jitter diretamente.

Segundo Kurose e Ross (2020), o Token Bucket é eficaz no controle de tráfego, mas deve ser complementado por outras técnicas para garantir QoS estrito, especialmente em redes com requisitos rígidos.

Comparação entre Token Bucket e Leaky Bucket

Característica	Token Bucket	Leaky Bucket
Permite bursts	Sim (até a capacidade do bucket)	Não (fluxo constante)
Controle de taxa média	Sim	Sim
Complexidade	Moderada	Baixa
Uso comum	Controle de tráfego, QoS, limitação em sistemas	Controle de tráfego, suavização de tráfego

Detalhamento Matemático do Algoritmo Token Bucket

Variáveis e Parâmetros

- r: taxa de geração de tokens (tokens por segundo)

- b: capacidade máxima do bucket (número máximo de tokens que podem ser acumulados)
- $T(t)$: número de tokens disponíveis no instante t
- $A(t_1, t_2)$: quantidade de dados (em bytes ou unidades) enviada entre os instantes t_1 e t_2
- C: capacidade máxima do link ou taxa máxima permitida (não necessariamente igual a r)

Modelo de geração de tokens

Os tokens são adicionados ao bucket a uma taxa constante r. A capacidade do bucket é limitada a b. Formalmente, para um intervalo de tempo $\Delta t = t - t_0$, a quantidade de tokens $T(t)$ cresce conforme:

$$
T(t) = \min \left(b, ; T(t_0) + r \times (t - t_0) \right)
$$

Ou seja, a quantidade de tokens no tempo t é o valor anterior $T(t_0)$ acrescido dos tokens gerados no intervalo $(t - t_0)$, limitado pelo máximo b.

Condição para envio de dados

Para que uma aplicação possa enviar um dado de tamanho s (em bytes ou unidades equivalentes), é necessário que existam tokens suficientes no bucket:

$$
T(t) \geq s
$$

Quando os dados são enviados, o número de tokens disponíveis diminui:

$$
T(t) \leftarrow T(t) - s
$$

Caso não haja tokens suficientes, o envio é bloqueado até que tokens sejam acumulados ou os dados são descartados, dependendo da política do sistema.

Controle da taxa de saída

O Token Bucket garante que, para qualquer intervalo de tempo $[t_1, t_2]$, a quantidade de dados $A(t_1, t_2)$ transmitidos satisfaça a desigualdade:

$$
A(t_1, t_2) \leq r \times (t_2 - t_1) + b
$$

Ou seja, a saída pode exceder a taxa r em até b unidades, o que representa o burst permitido pelo bucket.

Interpretação:

- A parte $r \times (t_2 - t_1)$ é o número de tokens gerados no intervalo.
- O termo b representa a "reserva" acumulada no bucket, permitindo bursts.

Taxa média

Se considerarmos um intervalo suficientemente longo, o throughput médio estará limitado a r:

$$
\lim_{t_2 - t_1 \to \infty} \frac{A(t_1, t_2)}{t_2 - t_1} \leq r
$$

Burst máximo permitido

O burst máximo $B_{\max}$ corresponde à capacidade b do bucket, ou seja, a quantidade máxima de dados que pode ser enviada de forma imediata (rajada) caso o bucket esteja cheio:

$$
B_{\max} = b
$$

Esse burst permite flexibilidade para que aplicações que geram tráfego de forma intermitente possam enviar dados rapidamente, sem ultrapassar o limite médio.

Exemplo Numérico

Suponha:

- $r = 1000$ tokens/segundo (permitindo enviar 1000 bytes por segundo)
- $b = 5000$ tokens (bucket que armazena até 5 KB)
- No instante $t_0 = 0$, o bucket está cheio $T(0) = 5000$.

Se um burst de 4000 bytes for enviado imediatamente, o bucket é reduzido:

$$
T(0^+) = 5000 - 4000 = 1000
$$

Se após 1 segundo, nenhum dado for enviado, o bucket acumulará tokens:

$$
T(1) = \min(5000, 1000 + 1000 \times 1) = 2000
$$

Isso permite novos bursts desde que $T(t) \geq s$.

Relação com outras abordagens e propriedades matemáticas

- O Token Bucket pode ser interpretado como um processo **"caixa de armazenamento"** limitado, com entrada constante (tokens) e saída regulada (dados).
- Pode ser formalizado com sistemas de filas e teoria de controle estocástico.
- Conforme descrito por Parekh e Gallager (1993), a relação entre o Token Bucket e o modelo de redes de filas ajuda a garantir garantias probabilísticas para atrasos e perdas, principalmente em redes com QoS.

Aplicações do Algoritmo Token Bucket no Dia a Dia e ao Longo dos Anos

1. Controle de tráfego em redes de computadores

A aplicação mais clássica e difundida do Token Bucket está no **controle de tráfego em redes IP**, especialmente em protocolos e dispositivos que precisam garantir qualidade de serviço (QoS).

- **Provedores de Internet (ISPs):** Utilizam Token Bucket para limitar a banda que cada usuário pode consumir, prevenindo que um cliente consuma todo o link disponível e prejudique os demais.
- **Roteadores e switches:** Implementam o algoritmo para controlar o fluxo de pacotes, permitindo bursts de dados, mas limitando a taxa média para evitar congestionamentos.
- **Redes corporativas e data centers:** Controlam o uso da largura de banda para diferentes aplicações (voz, vídeo, dados), garantindo que aplicações sensíveis a atraso recebam prioridade, enquanto outras são limitadas.

2. Sistemas operacionais e gerenciamento de recursos

Fora das redes, o Token Bucket é usado para **limitar o uso de recursos computacionais**, como CPU, disco ou acesso a APIs:

- **Rate limiting em APIs Web:** Serviços online, como Google, Twitter, e APIs públicas usam o Token Bucket para limitar o número de requisições por usuário ou por IP, evitando abusos e ataques DoS.
- **Gerenciamento de processos:** Sistemas operacionais podem usar o Token Bucket para limitar o uso de CPU por processos ou grupos, garantindo que nenhum processo monopolize o sistema.

3. Aplicações multimídia e streaming

- **Streaming de vídeo e áudio:** Permite controlar o buffer de envio de dados, acomodando picos e garantindo a taxa média para reprodução contínua.
- **Telefonia IP e videoconferência:** Permite gerenciar rajadas de pacotes de áudio e vídeo sem ultrapassar limites da rede, reduzindo perdas e jitter.

4. Segurança e mitigação de ataques

- **Proteção contra ataques DDoS:** Firewalls e sistemas de detecção usam o Token Bucket para limitar o número de conexões ou pacotes que um IP pode enviar, controlando picos suspeitos e evitando sobrecarga.
- **Autenticação e controle de acesso:** Controlam tentativas de login e requisições em sistemas para evitar abusos.

Evolução e Importância ao Longo do Tempo

Desde sua formalização nos anos 1970 e 1980, o Token Bucket vem se consolidando como o principal algoritmo para controle de fluxo com suporte a bursts, contrastando com abordagens mais rígidas como o Leaky Bucket.

- **Décadas de 1980-1990:** Implementação em roteadores e protocolos pioneiros, como ATM (Asynchronous Transfer Mode) e redes frame relay, onde o controle eficiente da largura de banda era crítico.
- **Anos 2000:** Ampliação para controle de tráfego em redes IP modernas, serviços de streaming, e início da aplicação em sistemas distribuídos e APIs Web.
- **Hoje:** Onipresente em sistemas de rate limiting em nuvem, segurança cibernética, microserviços, e infraestruturas altamente dinâmicas, como redes definidas por software (SDN).

Exemplos reais

- **Linux Traffic Control (tc):** Utiliza Token Bucket para configurar regras de controle de tráfego em interfaces de rede.
- **Amazon API Gateway e AWS Lambda:** Aplicam Token Bucket para limitar chamadas e proteger serviços.
- **Cloudflare e Akamai:** Usam o algoritmo para mitigar ataques e controlar tráfego malicioso.

Implementação do Token Bucket em Python

```python
import time
from threading import Lock

class TokenBucket:
    def __init__(self, rate, capacity):
        """
        Inicializa o Token Bucket.

        :param rate: taxa de geração de tokens (tokens por segundo)
        :param capacity: capacidade máxima do bucket (máximo de tokens)
        """
        self.rate = rate
        self.capacity = capacity
        self.tokens = capacity  # Começa cheio
        self.timestamp = time.monotonic()
```

```
        self.lock = Lock()

    def _add_tokens(self):
        """
        Atualiza a quantidade de tokens no bucket baseado no tempo
passado.
        """
        now = time.monotonic()
        elapsed = now - self.timestamp
        self.timestamp = now

        added_tokens = elapsed * self.rate
        self.tokens = min(self.capacity, self.tokens + added_tokens)

    def consume(self, tokens):
        """
        Tenta consumir uma quantidade de tokens. Retorna True se
possível, False caso contrário.

        :param tokens: quantidade de tokens a consumir
        """
        with self.lock:
            self._add_tokens()
            if self.tokens >= tokens:
                self.tokens -= tokens
                return True
            else:
                return False

    def get_tokens(self):
        """
        Retorna a quantidade atual de tokens disponíveis (para debug).
        """
        with self.lock:
            self._add_tokens()
            return self.tokens


# Exemplo de uso
if __name__ == "__main__":
    tb = TokenBucket(rate=5, capacity=10)  # 5 tokens/segundo,
capacidade 10 tokens

    for i in range(20):
        if tb.consume(3):
            print(f"Pacote {i+1} enviado. Tokens restantes:
{tb.get_tokens():.2f}")
        else:
            print(f"Pacote {i+1} rejeitado. Tokens insuficientes:
{tb.get_tokens():.2f}")
        time.sleep(0.5)
```

Explicação Python

- Usamos `time.monotonic()` para medir intervalos de tempo seguros.
- Um `Lock` garante segurança em acesso concorrente.
- Método `_add_tokens()` atualiza tokens com base no tempo decorrido.
- `consume()` verifica se há tokens suficientes e os remove.
- No exemplo, tentamos enviar pacotes de 3 tokens a cada 0.5s.

Implementação do Token Bucket em C

```c
#include <stdio.h>
#include <time.h>
#include <unistd.h>  // Para usleep
#include <pthread.h>

typedef struct {
    double rate;        // tokens por segundo
    double capacity;    // capacidade máxima do bucket
    double tokens;      // tokens atuais
    struct timespec last_time;
    pthread_mutex_t lock;
} TokenBucket;

double time_diff(struct timespec *start, struct timespec *end) {
    return (end->tv_sec - start->tv_sec) + (end->tv_nsec - start-
>tv_nsec) / 1e9;
}

void token_bucket_init(TokenBucket *tb, double rate, double capacity) {
    tb->rate = rate;
    tb->capacity = capacity;
    tb->tokens = capacity;
    clock_gettime(CLOCK_MONOTONIC, &tb->last_time);
    pthread_mutex_init(&tb->lock, NULL);
}

void token_bucket_add_tokens(TokenBucket *tb) {
    struct timespec now;
    clock_gettime(CLOCK_MONOTONIC, &now);

    double elapsed = time_diff(&tb->last_time, &now);

    pthread_mutex_lock(&tb->lock);
    tb->last_time = now;

    tb->tokens += elapsed * tb->rate;
```

```c
    if (tb->tokens > tb->capacity) {
        tb->tokens = tb->capacity;
    }

    pthread_mutex_unlock(&tb->lock);
}

int token_bucket_consume(TokenBucket *tb, double tokens) {
    int allowed = 0;
    token_bucket_add_tokens(tb);

    pthread_mutex_lock(&tb->lock);
    if (tb->tokens >= tokens) {
        tb->tokens -= tokens;
        allowed = 1;
    }
    pthread_mutex_unlock(&tb->lock);
    return allowed;
}

double token_bucket_get_tokens(TokenBucket *tb) {
    token_bucket_add_tokens(tb);

    pthread_mutex_lock(&tb->lock);
    double current_tokens = tb->tokens;
    pthread_mutex_unlock(&tb->lock);

    return current_tokens;
}

// Exemplo de uso
int main() {
    TokenBucket tb;
    token_bucket_init(&tb, 5.0, 10.0); // 5 tokens/s, capacidade 10
tokens

    for (int i = 0; i < 20; i++) {
        if (token_bucket_consume(&tb, 3.0)) {
            printf("Pacote %d enviado. Tokens restantes: %.2f\n", i+1,
token_bucket_get_tokens(&tb));
        } else {
            printf("Pacote %d rejeitado. Tokens insuficientes: %.2f\n",
i+1, token_bucket_get_tokens(&tb));
        }
        usleep(500000); // 0.5 segundos
    }

    pthread_mutex_destroy(&tb.lock);
    return 0;
}
```

Explicação C

- Usa `clock_gettime(CLOCK_MONOTONIC, ...)` para medir tempo seguro.
- `pthread_mutex_t` para proteger o acesso concorrente.
- O método `token_bucket_add_tokens` atualiza tokens conforme tempo decorrido.
- `token_bucket_consume` tenta consumir tokens e retorna sucesso/falha.
- O loop no `main` tenta enviar pacotes a cada 0.5 segundos.

Conclusão

O Token Bucket é um algoritmo fundamental para controle de fluxo e limitação de taxa em redes de computadores, com aplicabilidade crescente em sistemas distribuídos e aplicações modernas. Surgiu para suprir as limitações do Leaky Bucket, especialmente em permitir bursts controlados, melhorando a utilização eficiente dos recursos.

Embora possua limitações, é amplamente adotado devido à sua simplicidade e eficácia, e continua sendo estudado e adaptado para ambientes atuais, como redes definidas por software (SDN) e nuvem.

O Token Bucket é um algoritmo extremamente versátil e robusto, que desde sua criação tem sido usado para garantir um uso justo e eficiente de recursos limitados, especialmente em redes e sistemas computacionais. Seu suporte a bursts controlados o torna ideal para aplicações que precisam equilibrar desempenho e controle, o que explica seu uso massivo e contínuo na indústria tecnológica.

Referências adicionais para estudo matemático

- Parekh, A. K., & Gallager, R. G. (1993). A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking*, 1(3), 344-357.
- Cruz, R. L. (1991). A calculus for network delay, Part I: Network elements in isolation. *IEEE Transactions on Information Theory*, 37(1), 114-131.
- Jacobson, V. (1988). Congestion avoidance and control. *ACM SIGCOMM Computer Communication Review*, 18(4), 314-329.
- Floyd, S., & Jacobson, V. (1993). Random Early Detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4), 397–413.
- Jacobson, V., Nichols, K., & Poduri, S. (1999). An Expedited Forwarding PHB. *RFC 2598*.
- Stallings, W. (2007). *Data and Computer Communications* (8th ed.). Prentice Hall.
- Kurose, J. F., & Ross, K. W. (2020). *Computer Networking: A Top-Down Approach* (8th ed.). Pearson.