**Documentação Geral do Sistema ARBIS em 28/04/2025**

**Introdução**

O sistema ARBIS MVD é um sistema modular e escalável para monitoramento, detecção, e execução de arbitragens triangulares de criptoativos em tempo real. Utiliza worker\_threads do Node.js e WebSockets para coletar cotações de cinco exchanges (Binance, Bybit, OKX, Bitpreco, Deribit), processar dados com baixa latência, e executar ordens com eficiência. O sistema opera em janelas específicas de alta frequência de oportunidades (ex.: horário de mercado de Nova York), com todos os workers (dados, execução, e coordenador) inicializados uma vez no início da janela e mantidos ativos, sem reinicialização. Ele identifica oportunidades viáveis com lucro mínimo (gatilho = 0.003) e liquidez mínima (100 USDT), registra resultados em logs estruturados (CSV e snapshots), and suporta expansão para 6-10 exchanges.

**Objetivo**

* Desenvolver um sistema robusto para detectar arbitragens triangulares em tempo real, com base em dados de mercado de múltiplas exchanges.
* Executar ordens (T1, T2, T3 ou apenas T1, T2 em alguns casos) com baixa latência, garantindo validação, monitoramento, e estabilidade.
* Escalar modularmente para novas exchanges, mantendo resiliência e eficiência, com logs estruturados para auditoria.

**Arquitetura**

O sistema é estruturado em três níveis de workers, comunicando-se via MessageChannel para baixa latência:

* **Main (Nível 1, mainheap.js)**: Inicializa workers de dados e o coordenador, coleta cotações de 15 workers (3 pares x 5 exchanges), processa dados com heaps (cotacoes, bests), detecta arbitragens, e gera a matriz de operações. Integração com o coordenador está pendente (divergência 11).
* **Workers de Dados (Nível 2)**: Um por par/exchange (15 no total: Binance, Bybit, OKX, Bitpreco, Deribit; pares ex.: BTCUSDT, BTCBRL, USDTBRL), coletam dados de mercado via WebSocket público e enviam ao main. Desenvolvidos, com adaptações pendentes.
* **Worker Coordenador (Nível 2, coordenador.js)**: Worker persistente que orquestra a arbitragem, distribuindo ordens (T1, T2, T3, ou apenas T1, T2 em alguns casos) para workers de execução e exibindo resultados. Simula a matriz de operações, aguardando integração com o main.
* **Workers de Execução (Nível 3)**: Um por exchange (Binance, Bybit, OKX), gerenciam WebSockets privados para postagem e monitoramento de ordens. Workers para Bitpreco e Deribit serão desenvolvidos na expansão.
* **Workers Opcionais (Nível 2)**: Ex.: workerSaldos para monitoramento de saldos, planejado para futura implementação.

Conexões WebSocket são mantidas ociosas com pings periódicos (15-25s), com reconexão automática (5s). A inicialização sequencial (Binance → OKX/Bybit) evita erros intermitentes devido à demora nas conexões da Binance.

**Componentes**

**Main (mainheap.js)**

* **Função**: Gerencia a coleta de cotações em tempo real, processa dados para detectar arbitragens triangulares, e prepara a matriz de operações para o coordenador.
* **Implementação**:
  + Executa 15 workers de dados (3 pares x 5 exchanges: Binance, Bybit, OKX, Bitpreco, Deribit), definidos por argumentos (cripto, dolar, moeda, ex.: btc usdt brl).
  + Pares: cripto+dolar (BTCUSDT), cripto+moeda (BTCBRL), dolar+moeda (USDTBRL).
  + Usa heaps (bidHeaps, askHeaps) para manter bests (melhores bid/ask por par), atualizados via updateBests.
  + Calcula arbitragens (vendeDomesticamente, compraDomesticamente) com gatilho de 0,3% e liquidez mínima de 100 USDT.
* **Estruturas de Dados**:
  + **exchanges**:
  + const exchanges = [
  + { exchange: 'Bybit', fees: 0.001, venderPode: true, comprarPode: true },
  + { exchange: 'Binance', fees: 0.001, venderPode: true, comprarPode: true },
  + { exchange: 'Deribit', fees: 0.001, venderPode: true, comprarPode: true },
  + { exchange: 'bitpreco', fees: 0.001, venderPode: true, comprarPode: true },
  + { exchange: 'OKX', fees: 0.001, venderPode: true, comprarPode: true }
  + ];
    - Lista estática com taxas e permissões, futuramente com saldos dinâmicos via SharedArrayBuffer.
  + **cotacoes**:
  + const cotacoes = {
  + "Binance": {
  + "BTCUSDT": { bid: "91967.25", bidAmount: "5.2198", bidTime: "2025-03-03 10:04:58.495", ask: "91967.26", askAmount: "0.6324", askTime: "2025-03-03 10:04:58.495" },
  + // ...
  + },
  + // ...
  + };
    - Buffer global com cotações por exchange/par, incluindo preços, quantidades, e tempos.
  + **bests**:
  + const bests = {
  + "BTCUSDT": {
  + "bid": { preco: 91967.25, amount: 5.2198, time: "2025-03-03 10:04:58.495", exchange: "Binance" },
  + "ask": { preco: 91959.73, amount: 0.0206, time: "2025-03-03 10:04:58.462", exchange: "Bybit" }
  + },
  + // ...
  + };
    - Melhores bid/ask por par, mantidos por heaps (bidHeaps max, askHeaps min).
* **Funções Principais**:
  + **formatTime(date)**: Formata timestamps com milissegundos (ex.: "2025-03-03 10:00:00.123").
  + **registrarNoLog(output)**: Grava snapshots em saida.txt e CSV em Arbs[cripto][dolar][moeda].txt.
  + **atribuirCotacaoExchange(exchange, par, bid, bidAmount, bidTime, ask, askAmount, askTime)**: Atualiza cotacoes, chama updateBests se bid/ask mudam.
  + **updateBests(exchange, par, bid, bidAmount, bidTime, ask, askAmount, askTime)**:
    - Insere/atualiza heaps, seleciona melhores bid/ask, dispara testaArbitragens se bests mudar e isArbitragePaused for false.
    - Ajustes (07/03/2025): Filtra quantidades válidas (> 0), reconstrói heaps se inconsistentes, usa valores padrão seguros (preco: 0 para bid, Infinity para ask), e atualiza timestamps com bidTime/askTime.
  + **testaArbitragens()**:
    - Calcula:
      * vendeDomesticamente (VD): bests[cripto+moeda].bid.preco / (bests[dolar+moeda].ask.preco \* bests[cripto+dolar].ask.preco) - 1.
      * compraDomesticamente (CD): bests[cripto+dolar].bid.preco \* bests[dolar+moeda].bid.preco / bests[cripto+moeda].ask.preco - 1.
    - Valida: VD/CD > 0.003, amount > 100 USDT (usando preço de BTCUSDT da operação correspondente).
    - Ações: Exibe operações no console, dispara som se válida, pausa 10s, grava CSV e snapshots.
  + **inicializarWorkers(codigosFormaPares)**: Cria 15 workers de dados.
  + **setInterval()**: Marca tempo no console a cada 30s.
* **Comunicação**:
  + Recebe cotações dos workers de dados, envia configurações via worker\_threads.
  + Futuramente, enviará a matriz de operações ao coordenador via parentPort/MessageChannel (divergência 11).
* **Estado Atual**: Funcional para detecção, com matriz simulada no coordenador. Aguardando integração e execução de trades.
* **Pendências**:
  + Integração com o coordenador (divergência 11).
  + Formato do CSV (divergência 5).
  + Saldos dinâmicos em exchanges.

**Observação Sobre Comunicações. 05/05/2025.**

Na integração do coordenador com o main, o mainheap\_fase3.js não recebia mensagens conexoesProntas enviadas pelo coordenador\_fase3.js, impedindo o envio das matrizes de operações (mockMatrix3, mockMatrix2). A saída mostrava o coordenador notificando "Todos os workers prontos", mas o mainheap não registrava logs correspondentes, sugerindo falha na comunicação. A causa provável era um erro na configuração ou transferência do MessagePort (coordToMain) do MessageChannel, possivelmente devido a problemas na inicialização do canal ou sobrecarga do event loop do mainheap.

**Solução Adotada:** Substituir o MessageChannel por parentPort para mensagens do coordenador\_fase3.js ao mainheap\_fase3.js (ex.: conexoesProntas, arbitrageResult) e usar Worker.postMessage para mensagens do mainheap ao coordenador (ex.: ping, executeArbitrage). O mainheap diferenciará mensagens de arbitragem via tipo (msg.type). A comunicação com workers de execução permanece com MessageChannel, pois está funcional.

**Como Funciona:**

* parentPort: No coordenador\_fase3.js, parentPort (fornecido por worker\_threads) permite enviar mensagens diretamente ao processo pai (mainheap) com parentPort.postMessage. O mainheap recebe essas mensagens via coordenador.on('message', ...), eliminando a necessidade de transferir um MessagePort.
* Worker.postMessage: No mainheap\_fase3.js, o objeto Worker (coordenador) permite enviar mensagens ao coordenador com coordenador.postMessage, recebidas no coordenador via parentPort.on('message', ...).
* MessageChannel: Um mecanismo do Node.js que cria dois MessagePorts (port1, port2) para comunicação bidirecional. Requer transferência explícita de um porto (ex.: transferList) e configuração manual, o que pode falhar se mal configurado. Usado no sistema atual para coordenador-workers, mas substituído por parentPort/Worker na comunicação mainheap-coordenador para maior simplicidade.
* Funcionamento: O coordenador envia conexoesProntas via parentPort quando workers de execução estão prontos. O mainheap processa essas mensagens, atualiza conexoesProntas, e envia matrizes via coordenador.postMessage após 1s (mockMatrix3) e 15s (mockMatrix2). O mainheap usa msg.type para distinguir mensagens, com logs robustos para depuração.

**Workers de Dados**

* **Função**: Coletam dados de mercado (bid, ask, bidAmount, askAmount, bidTime, askTime) via WebSocket público para 15 pares/exchanges.
* **Implementação**:
  + workerBinance.js, workerBybit.js, workerOKX.js, workerBitpreco.js, workerDeribit.js.
  + Cada worker gerencia um WebSocket para um par (ex.: BTCUSDT na Binance).
  + Enviam cotações ao main com timestamps (abs, t).
* **Estado Atual**: Funcionais, com adaptações pendentes (ex.: padronização de pares, reconexão com backoff).
* **Robustez**: Reconexão (5s), pings periódicos, logs detalhados.

**Worker Coordenador (coordenador.js)**

* **Função**: Worker persistente que orquestra a arbitragem, inicializando workers de execução, enviando ordens (T1, T2, T3, ou apenas T1, T2 em alguns casos), monitorando status, and exibindo resultados.
* **Funcionalidades**:
  + **Inicialização**: Sequencial (Binance → OKX/Bybit), mantendo ports e conexoesProntas sem reinicialização para estabilidade.
  + **Matriz de Operações**: Processa uma matriz de duas ou três ordens, onde a primeira é T1, a segunda T2, e a terceira (se presente) T3. Exemplo:
  + [
  + { exchange: 'Bybit', symbol: 'BTCUSDT', side: 'sell', type: 'limit', amount: '0.0001', price: '95000', timeInForce: 'IOC' },
  + { exchange: 'OKX', symbol: 'BTCBRL', side: 'buy', type: 'market', amount: '0.0001', price: null, timeInForce: null },
  + { exchange: 'Binance', symbol: 'USDTBRL', side: 'sell', type: 'market', amount: '10', price: null, timeInForce: null }
  + ]
    - Algumas arbitragens podem ter apenas duas operações (T1, T2), dependendo da estratégia ou oportunidade detectada.
    - Atualmente simulada, futuramente recebida do main.
  + **Fluxo**:
    - Envia T1 (exeTradePromise, LIMIT, IOC) com timeout de 3s.
    - Prossegue com T2 (e T3, se presente) (exeTradeNormal, MARKET) se T1 filled, aborta se rejected, cancelled, or timeout.
    - Validação: Workers verificam sintaxe/parâmetros (accepted/rejected).
    - Execução: Status live, filled, or cancelled.
  + **Monitoramento**: Recebe orderStatus via Postagem (accepted, rejected) and Ordens (live, filled, cancelled).
  + **Tabela de Resultados**: Exibe T, Exchange, Par, Side, Type, Amount, Askd.Price, Exec.Price, TIF, Result, Time(ms), Reason. Ajustada para suportar arbitragens com duas ou três operações.
  + **Logs**: Timestamps (abs, t), cores ANSI, formatação JSON. Logs em arquivo pendentes (divergência 11).
  + **Rastreamento de Tempo**: t0 (simulado, futuramente do main), t (abs - t0), abs (Date.now()).
* **Compatibilidade**:
  + Envia executeOrder { type: 'executeOrder', order, orderId, t0 }, recebe conexoesProntas, orderStatus.
  + Suporta canais: Binance (um WebSocket), OKX (dois canais), Bybit (dois WebSockets).
  + Filtra subscrição estática da OKX (BTC-USDT, BTC-BRL, USDT-BRL), a ser resolvida com mainheap.js.
* **Mapeamento de Campos**:

| **Campo** | **OKX** | **Bybit** | **Binance** |
| --- | --- | --- | --- |
| symbol | instId (BTC-USDT) | symbol (BTCUSDT) | symbol (BTCUSDT) |
| side | side (buy/sell) | side (Buy/Sell) | side (BUY/SELL) |
| type | ordType (market/ioc/limit) | orderType (Market/Limit) | type (MARKET/LIMIT) |
| amount | sz (BTC) | qty (BTC) | quantity (BTC) |
| price | px | price | price |
| timeInForce | Parte de ordType | timeInForce (IOC/GTC/FOK) | timeInForce (IOC/GTC/FOK) |
| tgtCcy | base\_ccy (mercado) | - | - |

* **Robustez**:
  + Tratamento de erros em mensagens/envio.
  + Timeout de 3s para T1.
  + Estado mantido sem reinicialização de ports/conexoesProntas.
* **Estado Atual**: Funcional, simulando matriz. Aguardando integração com main.

**Workers de Execução**

* **Implementação**:
  + workerExecBinance.js: WebSocket privado, HMAC-SHA256, um canal.
  + workerExecBybit.js: Dois WebSockets (tradeWs, orderWs), HMAC-SHA256.
  + workerExecOKX.js: WebSocket privado, HMAC-SHA256, dois canais (order, orders).
* **Função**: Postam/monitoram ordens, mapeiam campos, gerenciam orderIdMap.
* **Robustez**: Reconexão (5s), pings (15-25s), logs com abs/t.
* **Estado Atual**: Funcionais para Binance, Bybit, OKX. Bitpreco/Deribit pendentes.

**Dinâmica de Arbitragem**

* **Detecção**:
  + Main calcula:
    - vendeDomesticamente (VD): Vende cripto+moeda, compra dolar+moeda, compra cripto+dolar.
    - compraDomesticamente (CD): Compra cripto+moeda, vende dolar+moeda, vende cripto+dolar.
    - Fórmulas:
      * VD: bests[cripto+moeda].bid.preco / (bests[dolar+moeda].ask.preco \* bests[cripto+dolar].ask.preco) - 1.
      * CD: bests[cripto+dolar].bid.preco \* bests[dolar+moeda].bid.preco / bests[cripto+moeda].ask.preco - 1.
    - Valida: VD/CD > 0.003, amount > 100 USDT (usando preço de BTCUSDT da operação correspondente).
  + Ações: Exibe operações, dispara som, pausa 10s, grava CSV/snapshots.
* **Execução**:
  + Coordenador envia T1 (LIMIT, IOC), aguarda status (accepted, rejected, live, filled, cancelled).
  + Se T1 filled, envia T2 (e T3, se presente) (MARKET); se rejected, cancelled, or timeout (3s), aborta.
  + Algumas arbitragens podem ter apenas duas operações (T1, T2), dependendo da oportunidade detectada pelo main.
  + Operações:
    - VD:
      * T1: SELL, bests[cripto+moeda].bid (ex.: BTCBRL).
      * T2: BUY, bests[dolar+moeda].ask (ex.: USDTBRL).
      * T3 (se presente): BUY, bests[cripto+dolar].ask (ex.: BTCUSDT).
    - CD:
      * T1: BUY, bests[cripto+moeda].ask (ex.: BTCBRL).
      * T2: SELL, bests[dolar+moeda].bid (ex.: USDTBRL).
      * T3 (se presente): SELL, bests[cripto+dolar].bid (ex.: BTCUSDT).
* **Tabela de Resultados**:
  + Exibe após conclusão/aborto, suportando duas ou três operações:
  + [coordenador - Resultado] [abs=1745764097220 t=351] Tabela de Resultados:
  + T | Exchange | Par | Side | Type | Amount | Askd.Price | Exec.Price | TIF | Result | Time(ms) | Reason
  + ---+----------+---------+------+--------+--------+------------+------------+-----+-----------+----------+------------------------
  + T1 | Bybit | BTCUSDT | sell | limit | 0.0001 | 95000 | --- | IOC | cancelled | 350 | EC\_NoImmediateQtyToFill
  + T2 | OKX | BTCBRL | buy | market | 0.0001 | --- | --- | --- | aborted | 350 | T1 Failed
  + T3 | Binance | USDTBRL | sell | market | 10 | --- | --- | --- | aborted | 350 | T1 Failed

**Fluxo de Dados**

* **Entrada**: Argumentos (cripto, dolar, moeda) definem pares via codigosFormaPares (ex.: [["BTC", "USDT"], ["BTC", "BRL"], ["USDT", "BRL"]]).
* **Coleta**: Workers de dados conectam WebSockets, enviam cotações (bid, ask, bidAmount, askAmount, bidTime, askTime) ao main.
* **Processamento**:
  + atribuirCotacaoExchange atualiza cotacoes.
  + updateBests mantém bests com heaps.
  + testaArbitragens calcula VD/CD, valida amount.
* **Saída**:
  + Console: Marcações (30s), operações com aviso (ex.: "Arbitragem não disparada: Amount abaixo do mínimo").
  + saida.txt: Snapshots e cotações (30s).
  + Arbs[cripto][dolar][moeda].txt: CSV de arbitragens válidas.
  + Tabela de resultados (coordenador).
* **Futuro**: Matriz de operações passada ao coordenador.

**Expansão Futura**

* **Novas Exchanges**: Adicionar KuCoin, NovaDAX, etc., para 6-10 exchanges.
* **Roteiro**:
  1. Desenvolver código para WebSocket/canais.
  2. Testar com coordenador simulado/mainProxy.
  3. Integrar ao coordenador, ajustando mapeamento.
* **Workers Opcionais**: workerSaldos para saldos.
* **Saldos Dinâmicos**: Implementar em exchanges via SharedArrayBuffer.

**Lacunas e Recomendações**

* **Divergência 5 (Formato do CSV)**:
  + Atual: CSV em Arbs[cripto][dolar][moeda].txt registra arbitragens válidas (detalhes não especificados).
  + Proposto: T,Exchange,Par,Side,Type,Amount,Askd.Price,Exec.Price,TIF,Result,Time\_ms,Reason, alinhado com a tabela de resultados.
  + Pendente: Definir formato final e implementar exportação.
* **Divergência 11 (Integração Main-Coordenador)**:
  + Substituir matriz simulada por comunicação via parentPort/MessageChannel.
  + Formato JSON (sem campo trade, ordem implica T1, T2, T3):
  + [
  + { exchange: 'Bybit', symbol: 'BTCUSDT', side: 'sell', type: 'limit', amount: '0.0001', price: '95000', timeInForce: 'IOC' },
  + { exchange: 'OKX', symbol: 'BTCBRL', side: 'buy', type: 'market', amount: '0.0001', price: null, timeInForce: null },
  + { exchange: 'Binance', symbol: 'USDTBRL', side: 'sell', type: 'market', amount: '10', price: null, timeInForce: null }
  + ]
* **Outras**:
  + Validação de instId/symbol no coordenador.
  + Timeout configurável (3s).
  + Logs em arquivo para auditoria.
  + Subscrição dinâmica da OKX via mainheap.js (criptoDaArbitragem, etc.).
  + Reconstrução de heaps no mainheap.js para consistência contínua.
  + Reconexão com backoff nos workers de dados.

**Notas Operacionais**

* **Janela Operacional**: Momentos de alta frequência (ex.: mercado de NY), com workers/coordenador persistentes.
* **Inicialização Sequencial**: Binance → OKX/Bybit, evitando erros de conexão.
* **Estabilidade**: Manter ports/conexoesProntas sem reinicialização.
* **Estado (28/04/2025)**: Main, workers de dados, coordenador, e workers de execução (Binance, Bybit, OKX) funcionais. Aguardando integração, logs em arquivo, e expansão.

**Etapa Atual do Desenvolvimento em 30/04/2025.**

Integracao do Coordenador ao Main (mainheap.js)