Tài Liệu Luồng Xử Lý Hệ Thống Tracking

Tổng Quan

Hệ thống tracking này được thiết kế để thu thập, xử lý và lưu trữ dữ liệu vị trí của tàu biển (vessel) và máy bay (aircraft) từ nhiều nguồn khác nhau. Hệ thống sử dụng kiến trúc **Data Fusion Pipeline** để hợp nhất dữ liệu từ nhiều nguồn, loại bỏ trùng lặp, và chọn ra dữ liệu chất lượng tốt nhất để xuất bản.

Nguồn Dữ Liệu Hiện Tại

Đang hoạt động:

- **AIS SignalR**: Nguồn chính đang chạy realtime cho vessel data

Đã chuẩn bị sẵn (chưa kích hoạt):

- **Data Fetcher Service**: Có thể fetch từ các API bên ngoài như:
  - Marine Traffic
  - Vessel Finder
  - China Port
  - Custom sources
  - ADSB Exchange (aircraft)
  - OpenSky (aircraft)

Kiến trúc Multi-Source Ready: Hệ thống đã được thiết kế sẵn sàng để tích hợp nhiều nguồn, chỉ cần kích hoạt và cấu hình.

Luồng Xử Lý Chính

1. Thu Thập Dữ Liệu - AIS SignalR Service

1.1 Kết Nối SignalR

Service này kết nối tới máy chủ AIS qua SignalR WebSocket và lắng nghe dữ liệu AIS thô.

File: `src/ais/ais-signalr.service.ts:101-256`

```
async connect() {
  const hubUrl =
    `${this.cfg.AIS_HOST}/api/signalR?` +
    new URLSearchParams({
      Device: this.cfg.AIS_DEVICE,
      ConnectionId: '',
      ActionTypeValue: this.cfg.AIS_ACTION_TYPE,
      Query: this.cfg.AIS_QUERY,
      UserId: String(this.cfg.AIS_USER_ID),
      IsQueryLastestDataBeforeStream: this.cfg.AIS_QUERY_LATEST_BEFORE_STREAM,
    }).toString();

  this.connection = new HubConnectionBuilder()
    .withUrl(hubUrl, { transport: HttpTransportType.WebSockets })
    .withAutomaticReconnect([1000, 2000, 5000, 10000])
    .configureLogging(LogLevel.Information)
    .build();

  // Đăng ký các event handlers
  this.connection.on('QueryCount', (count: number) => {
    this.start$.next({ state: QueryResultState.Start, count });
  });

  this.connection.on('QueryData', (data: AisModel[]) => {
    this.data$.next({ state: QueryResultState.Query, data: data ?? [] });
  });

  this.connection.on('QueryEnd', () => {
    this.end$.next({ state: QueryResultState.End });
  });

  await this.connection.start();
  this.setupAutoTrigger();
}
```

1.2 Query Động và Auto Trigger

Hệ thống hỗ trợ 2 chế độ query:

- **Static Query**: Query cố định từ config
- **Dynamic Query**: Query tự động tạo theo khoảng thời gian, hỗ trợ incremental mode

File: `src/ais/ais-signalr.service.ts:72-91`

```
private buildDynamicQuery(): string {
  const minutes = this.cfg.AIS_QUERY_MINUTES || 10;
  let lower: Date;

  // Incremental mode: tiếp tục từ lần query trước
  if (this.cfg.AIS_QUERY_INCREMENTAL && this.lastLowerBound) {
    lower = this.lastLowerBound;
  } else {
    // Lookback mode: quay lại N phút từ hiện tại
    lower = new Date(Date.now() - minutes * 60 * 1000);
    this.lastLowerBound = lower;
  }

  lower.setSeconds(0, 0);
  const yyyy = lower.getUTCFullYear();
  const mm = lower.getUTCMonth() + 1;
  const dd = lower.getUTCDate();
  const HH = lower.getUTCHours();
  const MM = lower.getUTCMinutes();

  // Tạo query động: updatetime >= DateTime(...)
  return `(updatetime >= DateTime(${yyyy}, ${mm}, ${dd}, ${HH}, ${MM}, 0))`;
}
```

Auto trigger được thiết lập để tự động gọi query theo interval:

File: `src/ais/ais-signalr.service.ts:331-364`

```
private setupAutoTrigger() {
  if (!this.cfg.AIS_AUTO_TRIGGER) return;

  const interval = this.cfg.AIS_AUTO_TRIGGER_INTERVAL_MS || 15000;

  // Trigger lần đầu ngay lập tức
  if (!this.triggering && this.connection?.connectionId) {
    this.triggering = true;
    this.triggerQuery({ usingLastUpdateTime })
      .finally(() => (this.triggering = false));
  }

  // Thiết lập interval tự động
  this.autoTimer = setInterval(async () => {
    if (this.triggering) return;
    if (!this.connection || !this.connection.connectionId) return;
    try {
      this.triggering = true;
      await this.triggerQuery({ usingLastUpdateTime });
    } finally {
      this.triggering = false;
    }
  }, interval);
}
```

2. Điều Phối Dữ Liệu - AIS Orchestrator Service

2.1 Subscribe và Normalize

Orchestrator subscribe vào data stream từ SignalR service và thực hiện normalize.

File: `src/ais/ais-orchestrator.service.ts:62-74`

```
onModuleInit() {
  this.logger.log('AIS Orchestrator starting...');
  // Subscribe to raw AIS data stream
  this.sub = this.aisSignalr.dataStream$.subscribe({
    next: ({ data }) => this.ingestBatch(data),
    error: (e) => this.logger.error('AIS raw stream error: ' + e.message),
  });
}
```

2.2 Xử Lý Batch

Mỗi batch dữ liệu AIS được xử lý qua các bước sau:

File: `src/ais/ais-orchestrator.service.ts:79-131`

```
private async ingestBatch(rows: AisModel[]) {
  if (this.disposed) return;
  const now = Date.now();

  try {
    this.stats.batches++;
    this.stats.rawRows += rows.length;

    // Bước 1: Normalize raw AIS to standard vessel format
    const normalized: NormVesselMsg[] = [];
    for (const raw of rows) {
      const msg = normalizeAis(raw);  // Gọi hàm normalize
      if (msg) normalized.push(msg);
    }

    this.stats.normalized += normalized.length;

    if (normalized.length === 0) return;

    // Bước 2: Ingest into fusion service
    this.vesselFusion.ingest(normalized, now);

    // Bước 3: Collect unique vessel keys
    const keys = new Set<string>();
    for (const msg of normalized) {
      const key = keyOfVessel(msg);  // MMSI hoặc IMO hoặc callsign
      if (key) keys.add(key);
    }

    // Bước 4: Make fusion decisions and publish
    for (const key of keys) {
      await this.processFusion(key, now);
    }
  } catch (e: any) {
    this.logger.error('ingestBatch failed: ' + e.message);
  }
}
```

3. Chuẩn Hóa Dữ Liệu - Normalizers

3.1 Normalize AIS Messages

Hàm `normalizeAis` chuẩn hóa dữ liệu AIS thô từ SignalR về format chuẩn `NormVesselMsg`.

File: `src/fusion/normalizers.ts:37-117`

```
export function normalizeAis(raw: any): NormVesselMsg | undefined {
  // Bước 1: Extract MMSI (bắt buộc)
  let rawMmsi: any =
    raw.mmsi ??
```

```
  raw.MMSI ??
  raw.Mmsi ??
  raw.shipMMSI ??
  raw.ShipMMSI;
if (rawMmsi == null) return undefined;

if (typeof rawMmsi === 'number') rawMmsi = String(rawMmsi);
if (typeof rawMmsi !== 'string') return undefined;
const mmsi = rawMmsi.trim();

// Validate MMSI format (5-10 digits)
if (!/^[0-9]{5,10}$/.test(mmsi)) return undefined;

// Bước 2: Extract Lat/Lon (bắt buộc)
let lat: any =
  raw.lat ??
  raw.Lat ??
  raw.LAT ??
  raw.latitude ??
  raw.Latitude ??
  raw.LATITUDE;
let lon: any =
  raw.lon ??
  raw.Lon ??
  raw.LON ??
  raw.longitude ??
  raw.Longitude ??
  raw.LONGITUDE ??
  raw.long ??
  raw.Long;

// Parse strings to numbers if needed
if (typeof lat === 'string') {
  const p = parseFloat(lat);
  if (!Number.isNaN(p)) lat = p;
}
if (typeof lon === 'string') {
  const p = parseFloat(lon);
  if (!Number.isNaN(p)) lon = p;
}

// Validate coordinates
if (typeof lat !== 'number' || typeof lon !== 'number') return undefined;
if (Math.abs(lat) > 90 || Math.abs(lon) > 180) return undefined;

// Bước 3: Extract timestamp (bắt buộc)
const tsIso =
  raw.updatetime ??
  raw.updateTime ??
  raw.UpdateTime ??
  raw.update_time ??
  raw.Update_Time ??
  raw.timestamp ??
  raw.ts;
const ts = toIsoUtc(tsIso);
if (!ts) return undefined;

// Bước 4: Extract các trường optional
const speed = raw.speed ?? raw.SOG ?? raw.Speed ?? raw.SPEED ?? raw.sog;
const course = raw.course ?? raw.COG ?? raw.Course ?? raw.COURSE ?? raw.cog;
const heading = raw.heading ?? raw.Heading ?? raw.HEADING;
const callsign = raw.callsign ?? raw.CallSign ?? raw.CALLSIGN ?? raw.call_sign;
const name = raw.name ?? raw.Name ?? raw.NAME ?? raw.shipName ?? raw.ShipName;
const imo = raw.imo ?? raw.IMO ?? raw.Imo;
const status = raw.status ?? raw.Status ?? raw.STATUS ?? raw.nav_status ?? raw.navStatus;
```

```
  // Bước 5: Tạo normalized message
  const m: NormVesselMsg = {
    source: 'ais',
    ts,
    mmsi,
    imo: sanitizeStr(imo),
    callsign: sanitizeStr(callsign),
    name: sanitizeStr(name),
    lat,
    lon,
    speed: numOrUndef(speed),
    course: numOrUndef(course),
    heading: numOrUndef(heading),
    status: sanitizeStr(status),
  };
  return m;
}
```

3.2 Format Chuẩn - NormVesselMsg

File: `src/fusion/types.ts:4-17`

```
export type NormVesselMsg = {
  source: VesselSource;  // 'ais' | 'marine_traffic' | 'vessel_finder' | ...
  ts: string;            // ISO-8601 UTC timestamp
  mmsi?: string;         // Maritime Mobile Service Identity
  imo?: string;          // International Maritime Organization number
  callsign?: string;     // Radio callsign
  name?: string;         // Vessel name
  lat: number;           // Latitude (required)
  lon: number;           // Longitude (required)
  speed?: number;        // Speed in knots
  course?: number;       // Course over ground in degrees
  heading?: number;      // True heading in degrees
  status?: string;       // Navigation status
};
```

4. Fusion Pipeline - Xử Lý Hợp Nhất Dữ Liệu

4.1 Cấu Hình Fusion

File: `src/fusion/config.ts:1-18`

```
export const FUSION_CONFIG = {
  WINDOW_MS: 5 * 60 * 1000,              // Cửa sổ thời gian 5 phút
  ALLOWED_LATENESS_MS: 10 * 60 * 1000,   // Cho phép muộn tối đa 10 phút
  MAX_EVENT_AGE_MS: 24 * 60 * 60 * 1000, // Loại bỏ event cũ hơn 24 giờ
  SPEED_LIMIT_KN: 60,                     // Giới hạn tốc độ hợp lý cho tàu
  ALPHA: 0.25,                            // Tham số cho thuật toán
  BETA: 0.08,                             // Tham số cho thuật toán
} as const;

export const SOURCE_WEIGHT = {
  marine_traffic: 0.9,   // Độ tin cậy cao nhất
  vessel_finder: 0.85,
  china_port: 0.8,
  custom: 0.7,
  default: 0.7,
  adsb_exchange: 0.9,    // Cho aircraft
  opensky: 0.85,
} as const;
```

4.2 Event Time Window Store

Class này lưu trữ các message trong một cửa sổ thời gian (sliding window) và tự động loại bỏ dữ liệu cũ.

File: `src/fusion/window-store.ts:3-38`

```
export class EventTimeWindowStore<T extends { ts: string }> {
  private readonly windows = new Map<string, T[]>();
  private readonly lastPublished = new Map<string, string>();

  constructor(private readonly windowMs = FUSION_CONFIG.WINDOW_MS) {}

  push(key: string, message: T, now = Date.now()): void {
    const list = this.windows.get(key) ?? [];
    list.push(message);

    // Tự động loại bỏ các message cũ hơn cửa sổ thời gian
    this.pruneList(list, now - this.windowMs);
    this.windows.set(key, list);
  }

  getWindow(key: string): T[] {
    return this.windows.get(key) ?? [];
  }

  setLastPublished(key: string, tsIso: string): void {
    this.lastPublished.set(key, tsIso);
  }

  getLastPublished(key: string): string | undefined {
    return this.lastPublished.get(key);
  }

  private pruneList(list: T[], thresholdMs: number): void {
    // Loại bỏ các element có event time < threshold
    let i = 0;
    for (const m of list) {
      const t = Date.parse(m.ts);
      if (Number.isFinite(t) && t >= thresholdMs) break;
      i++;
    }
    if (i > 0) list.splice(0, i);
  }
}
```

4.3 Thuật Toán Scoring

Hệ thống sử dụng thuật toán scoring đa tiêu chí để đánh giá chất lượng của mỗi message.

File: `src/fusion/utils.ts:44-78`

```
// 1. Source Score - Đánh giá độ tin cậy nguồn
export function scoreBySource(source?: string): number {
  if (!source) return SOURCE_WEIGHT.default;
  return (SOURCE_WEIGHT as any)[source] ?? SOURCE_WEIGHT.default;
}

// 2. Recency Score - Đánh giá độ mới của dữ liệu
export function recencyScore(tsIso: string, now = Date.now()): number {
  const t = parseIso(tsIso);
  if (!t) return 0;

  const ageMin = (now - t) / 60000;  // Tuổi của dữ liệu tính bằng phút
  const recency = Math.max(0, 1 - ageMin / 15);  // Giảm tuyến tính trong 15 phút
  return recency;
}

// 3. Physical Validity - Kiểm tra tính hợp lý về mặt vật lý
export function physicalValidityVessel(m: NormVesselMsg): 0 | 1 {
  return saneVessel(m) ? 1 : 0;
}

export function saneVessel(m: NormVesselMsg, now = Date.now()): boolean {
  const t = parseIso(m.ts);

  // Loại bỏ event quá cũ hoặc quá mới (có thể do sai clock)
  if (!t || Math.abs(now - t) > FUSION_CONFIG.MAX_EVENT_AGE_MS) return false;

  // Kiểm tra tọa độ hợp lệ
  if (!isFiniteNumber(m.lat) || !isFiniteNumber(m.lon)) return false;
  if (m.lat < -90 || m.lat > 90 || m.lon < -180 || m.lon > 180) return false;

  // Kiểm tra tốc độ hợp lý (tàu không thể đi quá nhanh)
  if (isFiniteNumber(m.speed) && m.speed! > FUSION_CONFIG.SPEED_LIMIT_KN * 1.5) return false;

  return true;
}

// 4. Tổng Hợp Score - Weighted combination
export function scoreVessel(m: NormVesselMsg, now = Date.now()): number {
  const recency = recencyScore(m.ts, now);
  const sw = scoreBySource(m.source);
  const pv = physicalValidityVessel(m);

  // Công thức tính điểm: 50% recency + 30% source weight + 20% validity
  return 0.5 * recency + 0.3 * sw + 0.2 * pv;
}
```

4.4 Vessel Fusion Service

Service này thực hiện logic fusion chính.

File: `src/fusion/vessel-fusion.service.ts:14-61`

```
@Injectable()
export class VesselFusionService {
  private readonly windows = new EventTimeWindowStore<NormVesselMsg>();
  private readonly lastPoint = new Map<string, { lat: number; lon: number; ts: number }>();

  constructor(private readonly lastPublishedStore: LastPublishedService) {}

  // Bước 1: Ingest messages vào window store
  ingest(messages: NormVesselMsg[], now = Date.now()): void {
    for (const m of messages) {
      if (!saneVessel(m, now)) continue;  // Bỏ qua message không hợp lệ
```

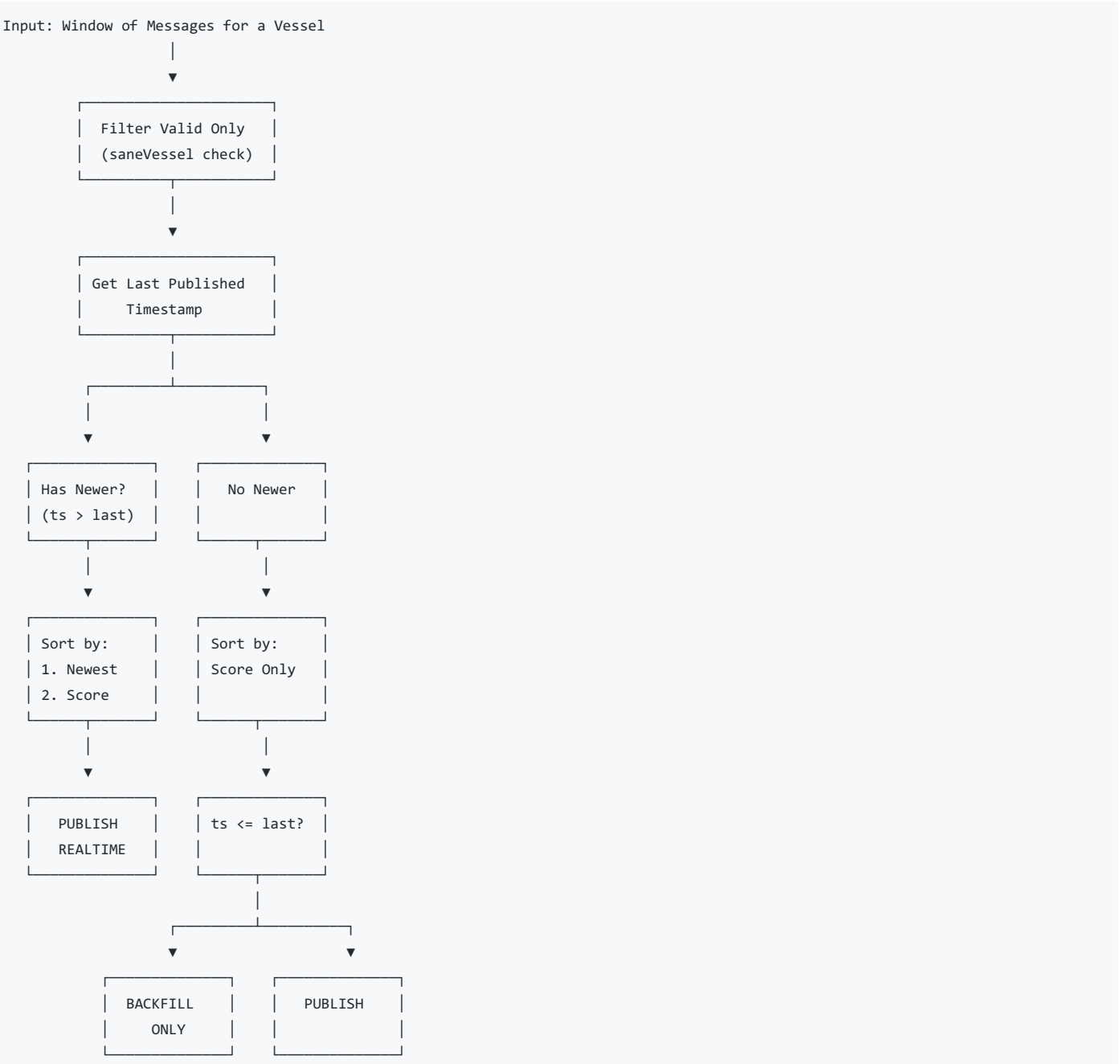
```
      const key = keyOfVessel(m);  // Extract key (MMSI hoặc IMO hoặc callsign)
      if (!key) continue;

      this.windows.push(key, m, now);  // Thêm vào window
    }
  }

  // Bước 2: Quyết định publish hay không
  async decide(key: string, now = Date.now()): Promise<FusionDecision<NormVesselMsg>> {
    // Lấy tất cả messages trong window cho vessel này
    const win = this.windows.getWindow(key).filter((m) => saneVessel(m, now));
    if (win.length === 0) return { publish: false, backfillOnly: false };

    // Lấy timestamp của lần publish cuối
    const last =
      (await this.lastPublishedStore.get('vessel', key)) ??
      this.windows.getLastPublished(key);

    // Tìm các message mới hơn lần publish cuối và không quá muộn
    const newer = win.filter(
      (m) =>
        (!last || Date.parse(m.ts) > Date.parse(last)) &&
        now - Date.parse(m.ts) <= FUSION_CONFIG.ALLOWED_LATENESS_MS,
    );

    let best: NormVesselMsg | undefined;

    if (newer.length > 0) {
      // Có message mới → chọn message tốt nhất để publish realtime
      best = newer.sort(
        (a, b) =>
          Date.parse(b.ts) - Date.parse(a.ts) ||  // Ưu tiên mới nhất
          scoreVessel(b, now) - scoreVessel(a, now)  // Nếu bằng nhau thì theo score
      )[0];
      return { best, publish: true, backfillOnly: false };
    }

    // Không có message mới → chọn message có score cao nhất
    best = win.sort((a, b) => scoreVessel(b, now) - scoreVessel(a, now))[0];

    if (best && last && Date.parse(best.ts) <= Date.parse(last)) {
      // Message này cũ hơn hoặc bằng lần publish cuối → chỉ backfill
      return { best, publish: false, backfillOnly: true };
    }

    return { best, publish: !!best, backfillOnly: false };
  }

  // Bước 3: Đánh dấu đã publish
  async markPublished(key: string, tsIso: string) {
    this.windows.setLastPublished(key, tsIso);
    await this.lastPublishedStore.set('vessel', key, tsIso);

    const t = Date.parse(tsIso);
    const win = this.windows.getWindow(key);
    const m = win.find((x) => Date.parse(x.ts) === t);
    if (m) this.lastPoint.set(key, { lat: m.lat, lon: m.lon, ts: t });
  }
}
```

4.5 Thuật Toán Quyết Định Publish

Thuật toán fusion hoạt động như sau:

1. **Filtering**: Lọc các message trong window, chỉ giữ lại message hợp lệ
2. **Deduplication**: Xác định các message "mới" (timestamp > last published)

3. **Selection**:
    - Nếu có message mới: chọn message mới nhất và có score cao nhất → **PUBLISH REALTIME**
    - Nếu không có message mới: chọn message có score cao nhất
        - Nếu timestamp <= last published → **BACKFILL ONLY** (chỉ lưu vào DB, không publish realtime)
        - Nếu timestamp > last published → **PUBLISH**



5. Xử Lý Fusion - Process Fusion

Sau khi fusion service đưa ra quyết định, orchestrator xử lý publish và persist.

File: `src/ais/ais-orchestrator.service.ts:136-157`

```
private async processFusion(key: string, now: number) {
  try {
    // Gọi fusion service để quyết định
    const decision = await this.vesselFusion.decide(key, now);

    if (!decision.best) return;

    // Publish realtime update nếu cần
    if (decision.publish) {
      const fused = this.toFusedRecord(decision.best);
      this.fused$.next(fused);  // Emit vào stream để SSE/WebSocket nhận
      this.stats.published++;
      await this.vesselFusion.markPublished(key, decision.best.ts);
    }

    // Persist to storage (cả realtime và backfill)
    if (decision.publish || decision.backfillOnly) {
      await this.persist(decision.best);
    }
  } catch (e: any) {
    this.logger.error(`processFusion failed for key ${key}: ${e.message}`);
  }
}
```

6. Persistence - Lưu Trữ Dữ Liệu

6.1 Redis Persistence

Dữ liệu được lưu vào Redis với 3 cấu trúc:

File: `src/ais/ais-orchestrator.service.ts:189-210`

```
const client = this.redis.getClient();

// 1. Geo Index - Cho spatial queries
await client.geoadd('ais:vessels:geo', msg.lon, msg.lat, mmsi);

// 2. Hash - Lưu thông tin chi tiết vessel
await client.hset(`ais:vessel:${mmsi}`, {
  lat: msg.lat.toString(),
  lon: msg.lon.toString(),
  ts: ts.toString(),
  speed: msg.speed?.toString() ?? '',
  course: msg.course?.toString() ?? '',
  heading: msg.heading?.toString() ?? '',
  status: msg.status ?? '',
  source: msg.source,
  mmsi,
  imo: msg.imo ?? '',
  callsign: msg.callsign ?? '',
  name: msg.name ?? '',
});

// 3. Sorted Set - Theo dõi vessel active (sorted by timestamp)
await client.zadd('ais:vessels:active', ts, mmsi);
```

Mục đích:

- `ais:vessels:geo` : Tìm kiếm vessels trong một vùng địa lý (bounding box, radius)
- `ais:vessel:{mmsi}` : Lấy thông tin nhanh của một vessel cụ thể
- `ais:vessels:active` : Lấy danh sách vessels active gần đây nhất

6.2 Postgres Persistence

Dữ liệu được lưu vào 2 bảng chính:

File: `src/ais/ais-orchestrator.service.ts:213-256`

```
await this.prisma.$transaction(async (tx) => {
  // Bảng 1: Vessel (master data)
  const vessel = await tx.vessel.upsert({
    where: { mmsi },
    create: {
      mmsi,
      vesselName: msg.name ?? undefined,
    },
    update: {
      vesselName: msg.name ?? undefined,
    },
  });

  // Bảng 2: VesselPosition (time-series data)
  await tx.vesselPosition.upsert({
    where: {
      vesselId_timestamp_source: {
        vesselId: vessel.id,
        timestamp: new Date(ts),
        source: msg.source ?? null,
      },
    },
    create: {
      vesselId: vessel.id,
      latitude: msg.lat,
      longitude: msg.lon,
      speed: msg.speed ?? null,
      course: msg.course ?? null,
      heading: msg.heading ?? null,
      status: msg.status ?? null,
      timestamp: new Date(ts),
      source: msg.source ?? null,
    },
    update: {
      latitude: msg.lat,
      longitude: msg.lon,
      speed: msg.speed ?? null,
      course: msg.course ?? null,
      heading: msg.heading ?? null,
      status: msg.status ?? null,
    },
  });
});
```

Composite Key: `(vesselId, timestamp, source)` đảm bảo không có duplicate và cho phép lưu cùng một thời điểm từ nhiều nguồn khác nhau.

7. API Endpoints

7.1 SSE Stream - Realtime Data

Client có thể subscribe vào SSE endpoint để nhận dữ liệu realtime.

File: `src/ais/ais.controller.ts:38-68`

```
@Sse('stream')
stream(
  @Query('trigger') trigger?: string,
  @Query('mode') mode?: string,
): Observable<MessageEvent> {

  // Heartbeat để giữ connection alive
  const heartbeat$ = interval(15000).pipe(
    map(() => ({ type: 'keepalive', data: null })),
    startWith({ type: 'keepalive', data: null }),
  );

  // Mode 'fused': chỉ nhận fused records
  if (mode === 'fused') {
    const fused$ = this.orchestrator.fusedStream$.pipe(
      map((rec) => ({ type: 'Fused', data: rec }))
    );
    return merge(heartbeat$, fused$) as unknown as Observable<MessageEvent>;
  }

  // Mode 'raw': nhận raw AIS data
  const start$ = this.ais.startStream$.pipe(
    map(({ state, count }) => ({ type: state, data: { count } })),
  );
  const data$ = this.ais.dataStream$.pipe(
    map(({ state, data }) => ({ type: state, data }))
  );
  const end$ = this.ais.endStream$.pipe(
    map(({ state }) => ({ type: state, data: null }))
  );

  return merge(heartbeat$, start$, data$, end$) as unknown as Observable<MessageEvent>;
}
```

Event Types:

- `keepalive` : Heartbeat mỗi 15s
- `Fused` : Fused vessel record (mode=fused)
- `Start` : Query bắt đầu với count
- `Query` : Batch dữ liệu AIS thô (mode=raw)
- `End` : Query kết thúc

7.2 Trigger Query

File: `src/ais/ais.controller.ts:71-83`

```
@Post('trigger')
async trigger(@Body() dto: QueryRequestDto) {
  try {
    const data = await this.ais.triggerQuery({
      query: dto?.query,
      usingLastUpdateTime: dto?.usingLastUpdateTime,
      userId: dto?.userId,
    });
    return { success: true, data };
  } catch (err: any) {
    throw new HttpException(`Trigger failed: ${err.message}`, 500);
  }
}
```

7.3 Status Endpoint

File: `src/ais/ais.controller.ts:92-101`

```
@Get('status')
async status() {
  const signalr = this.ais.getStatus();
  const orchestratorStats = this.orchestrator.getStats();
  return {
    signalr,
    orchestrator: orchestratorStats,
    now: new Date().toISOString(),
  };
}
```

8. Tóm Tắt Luồng Xử Lý

8.1 Luồng Dữ Liệu AIS

```
1. SignalR Hub → QueryData event
        ↓
2. AIS SignalR Service → data$ stream
        ↓
3. AIS Orchestrator → ingestBatch()
        ↓
4. Normalizers → normalizeAis()
        ↓  NormVesselMsg[]
5. Vessel Fusion Service → ingest() → push to window
        ↓
6. Vessel Fusion Service → decide() → apply scoring algorithm
        ↓
7. AIS Orchestrator → processFusion()
        ├─→ Publish to fused$ stream (SSE)
        └─→ Persist to Redis & Postgres
```

8.2 Key Points

1. **Event Time Processing**: Hệ thống xử lý theo event time (timestamp trong data), không phải processing time
2. **Sliding Window**: Duy trì window 5 phút, tự động loại bỏ dữ liệu cũ
3. **Multi-Source Fusion**: Hỗ trợ nhiều nguồn dữ liệu, mỗi nguồn có weight khác nhau
4. **Deduplication**: Tránh publish trùng lặp bằng cách theo dõi last published timestamp
5. **Late Data Handling**: Cho phép dữ liệu đến muộn tới 10 phút (ALLOWED_LATENESS_MS)
6. **Backfill Support**: Dữ liệu cũ vẫn được lưu vào DB nhưng không publish realtime
7. **Incremental Query**: Hỗ trợ query incremental để giảm load và tránh duplicate

9. Cấu Hình Hệ Thống

9.1 Environment Variables

```
# AIS SignalR Configuration
AIS_HOST=https://ais-server.example.com
AIS_DEVICE=device-id
AIS_ACTION_TYPE=1
AIS_USER_ID=12345
AIS_QUERY=(updatetime >= DateTime(2024, 1, 1, 0, 0, 0))
AIS_QUERY_LATEST_BEFORE_STREAM=false
AIS_USING_LAST_UPDATE_TIME=false

# Auto Trigger Configuration
AIS_AUTO_TRIGGER=true
AIS_AUTO_TRIGGER_INTERVAL_MS=15000

# Dynamic Query Configuration
AIS_QUERY_MINUTES=10
AIS_QUERY_INCREMENTAL=true

# Debug
AIS_DEBUG=false
AIS_QUERY_EVENT_TIMEOUT_MS=10000
```

9.2 Fusion Config

File: `src/fusion/config.ts`

```
WINDOW_MS: 5 * 60 * 1000              // 5 phút
ALLOWED_LATENESS_MS: 10 * 60 * 1000   // 10 phút
MAX_EVENT_AGE_MS: 24 * 60 * 60 * 1000 // 24 giờ
SPEED_LIMIT_KN: 60                     // 60 knots
```

10. Aircraft Fusion

Aircraft fusion hoạt động tương tự vessel fusion với một số điểm khác biệt:

File: `src/fusion/aircraft-fusion.service.ts:14-62`

- Sử dụng `NormAircraftMsg` thay vì `NormVesselMsg`
- Key là `icao24` hoặc `registration` hoặc `callsign`
- Speed limit khác (750 knots thay vì 60 knots)
- Thuật toán scoring tương tự nhưng với physical validity khác

```
export function saneAircraft(m: NormAircraftMsg, now = Date.now()): boolean {
  const t = parseIso(m.ts);
  if (!t || Math.abs(now - t) > FUSION_CONFIG.MAX_EVENT_AGE_MS) return false;
  if (!isFiniteNumber(m.lat) || !isFiniteNumber(m.lon)) return false;
  if (m.lat < -90 || m.lat > 90 || m.lon < -180 || m.lon > 180) return false;
  if (isFiniteNumber(m.groundSpeed) && m.groundSpeed! > 750) return false;
  return true;
}
```

11. Multi-Source Architecture - Kiến Trúc Đa Nguồn

11.1 Tình Trạng Hiện Tại

Hiện tại hệ thống **chỉ có AIS SignalR đang chạy thực tế**, nhưng đã được thiết kế sẵn sàng cho multi-source.

11.2 Data Fetcher Service (Sẵn Sàng Nhưng Chưa Kích Hoạt)

File: `src/data-fetcher/data-fetcher.service.ts:1-285`

Service này có thể fetch dữ liệu từ các API bên ngoài theo lịch trình (cron job).

```
@Injectable()
export class DataFetcherService {
  constructor(
    private prisma: PrismaService,
    private aircraftService: AircraftService,
    private vesselService: VesselService,
    private redisService: RedisService,
    private vesselFusion: VesselFusionService,     // Sử dụng cùng fusion service!
    private aircraftFusion: AircraftFusionService,
  ) {}

  // Hiện đang bị comment - có thể kích hoạt bất cứ lúc nào
  // @Cron(CronExpression.EVERY_10_SECONDS)
  async handleCron() {
    await this.fetchAndUpdateAircraftData();
    await this.fetchAndUpdateVesselData();
  }
}
```

11.3 Cách Thêm Nguồn Mới

Để thêm một nguồn dữ liệu mới (ví dụ: Marine Traffic API), làm theo các bước sau:

Bước 1: Fetch Data Từ API

```
// src/data-fetcher/data-fetcher.service.ts
private async fetchMarineTrafficData() {
  // Gọi Marine Traffic API
  const response = await axios.get('https://api.marinetraffic.com/vessels', {
    params: { apikey: this.config.MARINE_TRAFFIC_API_KEY }
  });

  const now = Date.now();

  // Normalize dữ liệu từ Marine Traffic format
  const normalized = response.data.map(raw =>
    normalizeVessel(
      {
        mmsi: raw.MMSI,
        lat: raw.LAT,
        lon: raw.LON,
        ts: raw.TIMESTAMP,
        speed: raw.SPEED,
        course: raw.COURSE,
        heading: raw.HEADING,
        name: raw.SHIPNAME,
        callsign: raw.CALLSIGN,
        status: raw.STATUS
      },
      'marine_traffic'  // Chỉ định source!
    )
  ).filter(m => m !== undefined);

  // Ingest vào CÙNG MỘT fusion service với AIS!
  this.vesselFusion.ingest(normalized, now);

  // Collect unique keys
  const keys = new Set<string>();
  for (const msg of normalized) {
    const key = keyOfVessel(msg);
    if (key) keys.add(key);
  }

  // Process fusion cho từng vessel
```

```
  for (const key of keys) {
    const decision = await this.vesselFusion.decide(key, now);

    if (decision.best) {
      // Upsert vessel
      const vessel = await this.vesselService.upsertVessel({
        mmsi: decision.best.mmsi,
        vesselName: decision.best.name,
      });

      // Persist position
      await this.vesselService.addPosition(vessel.id, {
        latitude: decision.best.lat,
        longitude: decision.best.lon,
        speed: decision.best.speed,
        course: decision.best.course,
        heading: decision.best.heading,
        status: decision.best.status,
        timestamp: new Date(decision.best.ts),
        source: decision.best.source,  // 'marine_traffic'
        score: scoreVessel(decision.best, now),
      });

      // Publish nếu cần
      if (decision.publish) {
        await this.redisService.publish('vessel:position:update',
          JSON.stringify({...decision.best})
        );
        await this.vesselFusion.markPublished(key, decision.best.ts);
      }
    }
  }
}
```

Bước 2: Đăng Ký Cron Job

```
// Fetch Marine Traffic data mỗi 30 giây
@Cron('*/30 * * * * *')
async fetchMarineTraffic() {
  await this.fetchMarineTrafficData();
}
```

Bước 3: (Tùy chọn) Điều Chỉnh Source Weight

Nếu muốn điều chỉnh độ tin cậy của nguồn:

```
// src/fusion/config.ts
export const SOURCE_WEIGHT = {
  marine_traffic: 0.95,  // Tăng weight nếu nguồn này tin cậy hơn
  vessel_finder: 0.85,
  ais: 0.80,             // AIS có thể có weight thấp hơn
  china_port: 0.8,
  custom: 0.7,
  default: 0.7,
} as const;
```

11.4 Cách Fusion Xử Lý Multi-Source

Khi có nhiều nguồn cùng báo cáo về một vessel, fusion service sẽ:

Ví dụ Thực Tế:

Giả sử vessel MMSI=123456789 có dữ liệu từ 3 nguồn trong cùng một window 5 phút:

```
Message 1 (AIS):
  ts: 2024-10-27T10:00:00Z
  lat: 10.5000, lon: 106.6000
  source: 'ais'
  source_weight: 0.8
  recency: 1.0 (mới nhất)
  score: 0.5×1.0 + 0.3×0.8 + 0.2×1.0 = 0.94

Message 2 (Marine Traffic):
  ts: 2024-10-27T09:58:00Z
  lat: 10.5001, lon: 106.6001
  source: 'marine_traffic'
  source_weight: 0.95
  recency: 0.87 (2 phút trước)
  score: 0.5×0.87 + 0.3×0.95 + 0.2×1.0 = 0.92

Message 3 (Vessel Finder):
  ts: 2024-10-27T09:55:00Z
  lat: 10.4999, lon: 106.5999
  source: 'vessel_finder'
  source_weight: 0.85
  recency: 0.67 (5 phút trước)
  score: 0.5×0.67 + 0.3×0.85 + 0.2×1.0 = 0.79
```

Quyết định:

1. Sort theo timestamp: Message 1 mới nhất
2. Nếu score gần bằng nhau → chọn message mới nhất
3. **Message 1 (AIS) được chọn** vì mới nhất và score cao nhất

Lợi ích:

- Tự động chọn data tốt nhất từ nhiều nguồn
- Không cần code riêng để merge
- Tránh duplicate
- Nguồn tin cậy hơn được ưu tiên khi timestamp gần bằng nhau

11.5 Persistence Multi-Source

Mỗi nguồn sẽ được lưu riêng trong database với composite key `(vesselId, timestamp, source)`:

```
-- VesselPosition table
vesselId | timestamp           | source            | lat     | lon      | speed
---------|---------------------|-------------------|---------|----------|------
123      | 2024-10-27 10:00:00 | ais               | 10.5000 | 106.6000 | 12.5
123      | 2024-10-27 09:58:00 | marine_traffic    | 10.5001 | 106.6001 | 12.3
123      | 2024-10-27 09:55:00 | vessel_finder     | 10.4999 | 106.5999 | 12.4
```

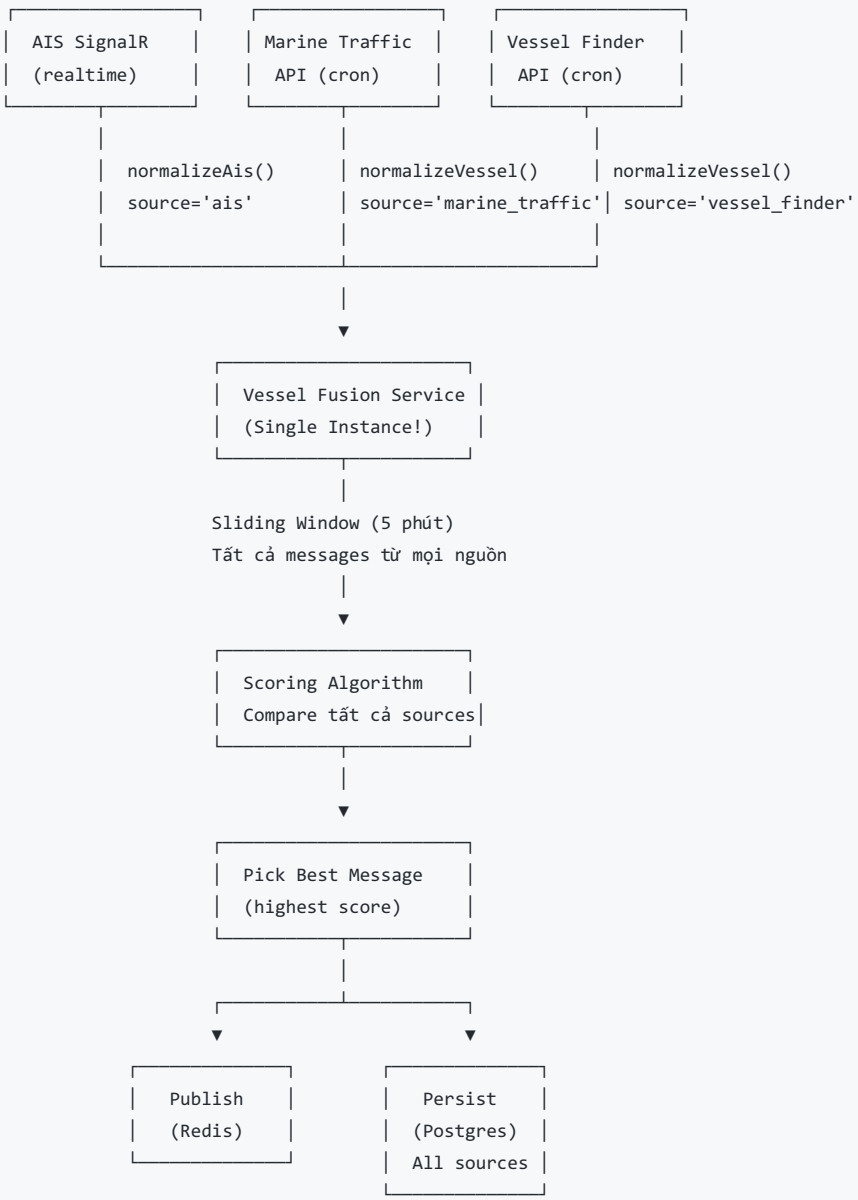
File: `src/ais/ais-orchestrator.service.ts:228-235`

```
await tx.vesselPosition.upsert({
  where: {
    vesselId_timestamp_source: {  // Composite key
      vesselId: vessel.id,
      timestamp: new Date(ts),
      source: msg.source ?? null,  // Mỗi source một record
    },
  },
  create: { /* ... */ },
  update: { /* ... */ },
});
```

Lợi ích:

- Giữ lại toàn bộ raw data từ mọi nguồn
- Có thể audit/debug khi cần
- Có thể re-run fusion với config khác
- Không mất dữ liệu khi thêm nguồn mới

11.6 Luồng Multi-Source



11.7 Kích Hoạt Data Fetcher

Để bắt đầu sử dụng multi-source, chỉ cần:

File: `src/data-fetcher/data-fetcher.service.ts:30`

```
// Bỏ comment dòng này:
@Cron(CronExpression.EVERY_10_SECONDS)
async handleCron() {
  // ...
}
```

Hoặc tùy chỉnh schedule:

```
@Cron('0 */5 * * * *')  // Mỗi 5 phút
async fetchVesselFinderData() { /* ... */ }

@Cron('0 */2 * * * *')  // Mỗi 2 phút
async fetchMarineTrafficData() { /* ... */ }

@Cron('*/30 * * * * *') // Mỗi 30 giây
async fetchADSBExchangeData() { /* ... */ }
```

Kết Luận

Hệ thống tracking này sử dụng kiến trúc data fusion pipeline với các thành phần:

1. **Data Ingestion**: SignalR WebSocket cho realtime data
2. **Normalization**: Chuẩn hóa dữ liệu từ nhiều format khác nhau
3. **Fusion**: Sliding window + scoring algorithm để chọn data tốt nhất
4. **Persistence**: Redis (fast access) + Postgres (historical data)
5. **Streaming**: SSE để push realtime updates cho clients

Thuật toán fusion đảm bảo:

- **Quality**: Chọn dữ liệu có score cao nhất (recency + source weight + validity)
- **Freshness**: Ưu tiên dữ liệu mới nhất trong window
- **Deduplication**: Không publish trùng lặp
- **Completeness**: Vẫn lưu backfill data vào DB